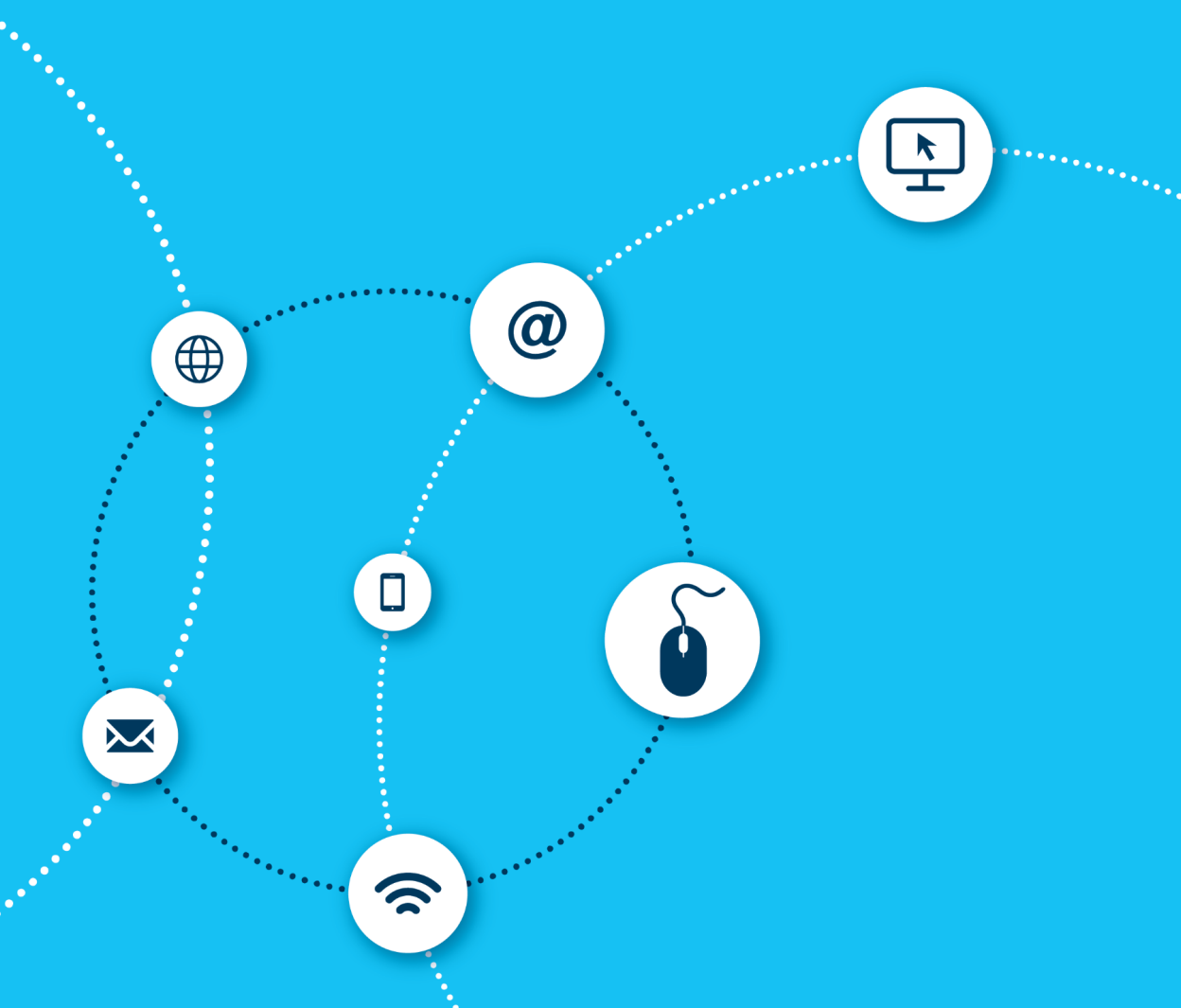
ECDL Romania

COMPUTING

Syllabus 1.0



ECDL – European Computer Driving Licence

ECDL COMPUTING

Competenţe în domeniul gândirii computaționale și programării

Editura ECDL ROMANIA
București, 2021

ISBN 978-606-9037-23-2

CUPRINS

CAPITOLUL 1 –
GÂNDEȘTE CA UN PROGRAMATOR

La finalul acestui capitol, veți putea să:

- Definiți termenul computing
- Definiți termenul de gândire computațională
- Identificați tehnici de gândire computațională precum descompunerea, recunoașterea de șabloane, abstractizarea și utilizarea algoritmilor
- Utilizați descompunerea unei probleme pentru a diviza date, procese sau probleme complexe în părți mai mici
- Definiți termenul de program
- Înțelegeți modul de utilizare al algoritmilor în gândirea computațională
- Identificați șabloanele în cadrul problemelor descompuse
- Utilizați abstractizarea pentru a filtra detaliile considerate inutile în analizarea unei probleme

1.1 GÂNDIREA COMPUTAȚIONALĂ

Concepte

Computing reprezintă efectuarea calculelor sau procesarea datelor, cu ajutorul unui computer.

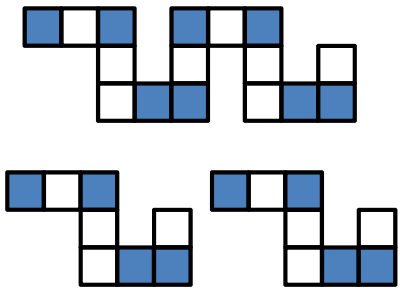
Gândirea computațională reprezintă procesul de analiză a problemelor și de identificare a soluțiilor posibile de rezolvare.

Gândirea computațională este utilă în rezolvarea problemelor și sarcinilor complexe precum proiectarea produselor, gătitul mâncărurilor, planificarea evenimentelor, repararea obiectelor defecte, asamblarea diverselor piese de mobilier, precum și în multe alte situații

Gândirea computațională utilizează 4 tehnici cheie de rezolvare a problemelor. Acestea pot fi utilizate în orice ordine și în orice combinație.

Recunoașterea de șabloane

Un șablon reprezintă un element sau o caracteristică repetitivă, cum ar fi de exemplu un motiv de pe o țesătură. Șabloanele pot fi regăsite de asemenea și în activități, cum ar fi diverse rețete ce pot implica setarea cuptorului la o anumită temperatură și așteptarea încălzirii cuptorului. Acest lucru este cunoscut sub denumirea de șablon partajat. Recunoașterea șabloanelor implică identificarea șabloanelor repetitive în cadrul problemelor complexe sau în cadrul problemelor mai mici, conexe.



În figura de mai sus, primul set de blocuri (pătrate) colorate a fost apoi separat în 2 porțiuni pentru a evidenția prezența unui șablon repetitiv.

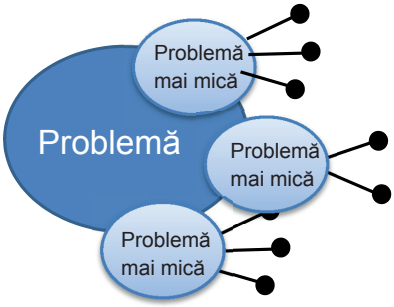
Abstractizarea

Abstractizarea reprezintă procesul de extragere a celor mai importante caracteristici din cadrul unei probleme sau sarcini. Caracteristicile extrase oferă informații utile în examinarea problemei și identificarea soluțiilor potențiale. Abstractizarea implică eliminarea detaliilor inutile și identificarea doar a informațiilor relevante pentru rezolvarea problemei. În cadrul unei sarcini de coacere a unor biscuiți, de exemplu, nu este important dacă persoana este dreptace sau stângace. Informațiile relevante în acest caz includ ingredientele folosite, ordinea de amestecare a lor, precum și durata și temperatura de coacere.

În exemplul de mai jos, din a doua imagine au fost eliminate toate detaliile inutile astfel încât să rămână doar informațiile relevante în rezolvarea problemei și anume că este vorba despre un câine care vrea mâncare.

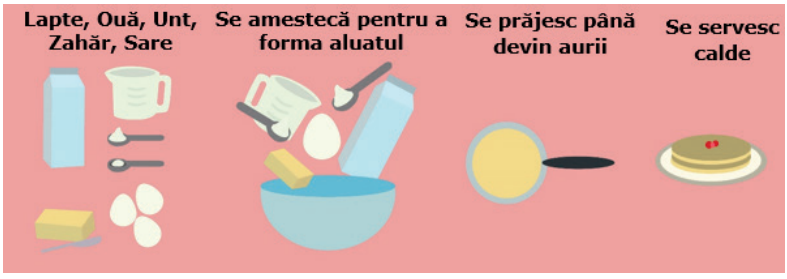
Descompunerea

Descompunerea implică divizarea unei probleme complexe în probleme mai mici și mai simple. Apoi, aceste probleme mai simple pot fi la rândul lor descompuse în probleme și mai mici, până când acestea devin ușor de înțeles și de gestionat. În cadrul unei sarcini de coacere a unor biscuiți de exemplu, o problemă minoră ar putea fi asigurarea faptului că cuptorul este setat la temperatura potrivită. Imaginea de mai jos ilustrează modul de descompunere a unei probleme în părți din ce în ce mai mici.



Algoritmi

Un algoritm reprezintă un set organizat de instrucțiuni ce oferă pașii de rezolvare a unei probleme sau sarcini. De exemplu, un algoritm ar putea consta în instrucțiunile dintr-o rețetă culinară sau calculele matematice pe care un computer trebuie să le urmeze. Elaborarea algoritmilor reprezintă procesul de creare a unor instrucțiuni bine definite, sub forma unor pași de urmat pentru a rezolva o problemă sau a îndeplini o sarcină cu succes. Un posibil algoritm pentru a face clătite implică parcurgerea următorilor pași:



Există două abordări suplimentare care includ adesea ca parte a gândirii computaționale evaluarea și generalizarea:

Evaluarea

Evaluarea implică validarea designului unui produs sau unui algoritm prin verificarea faptului că acesta funcționează conform specificațiilor sau că rezolvă problema în cauză.

Generalizarea

Generalizarea constă în identificarea unei modalități de a face o anumită soluție utilă în cadrul unui set mai larg de circumstanțe. Spre exemplu, ați putea folosi simboluri în loc de cuvinte la controlul produselor astfel încât această metodă să poată fi folosită, indiferent de limbă. În imaginea următoare, structura de bază a florii este întotdeauna aceeași și poate fi utilizată în mod repetat, variind însă caracteristici precum culoarea sau forma florii.

Aceste șase tehnici de gândire computațională pot fi folosite în paralel pentru a rezolva probleme complexe în domeniul informaticii și nu numai.

Pași

Exemplu: Proiectarea unei mașini de spălat rufe

În acest exemplu, tehnicile de gândire computațională sunt aplicate problemei complexe constând în proiectarea unei mașini de spălat rufe.

Abstractizare

Primul pas în proiectarea unei mașini de spălat rufe constă în determinarea scopului ei – spre exemplu, ar trebui să conțină programe de spălare intensivă sau delicată, la o temperatură mică sau mare, iar la finalul ciclului de spălare, rufele să fie curate. Faza de proiectare implică enumerarea caracteristicilor necesare pentru a atinge scopul propus. Abstractizarea poate fi folosită pentru a-l ajuta pe proiectant să filtreze informațiile relevante de cele nerelevante pentru a determina ce caracteristici să includă și ce anume să excludă. Detaliile nerelevante ar putea include culoarea mașinii de spălat sau dacă rufele ce urmează a fi spălate conțin 6 perechi de șosete sau 3 perechi de șosete.

Detaliile relevante sunt cele care afectează funcționalitatea generală și ar putea include informații precum temperatura apei sau dacă un anumit program este destinat spălării rufelor delicate sau țesăturilor rezistente la uzură. Aceste detalii sunt relevante în rezolvarea problemei de a obține la finalul ciclului de spălare rufe curate, fără a le deteriora.

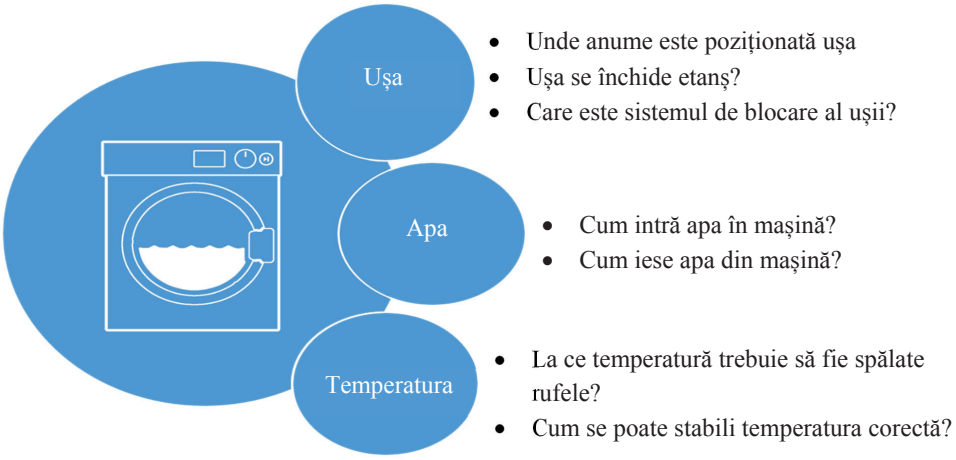
Descompunere

Descompunerea poate fi utilizată pentru a diviza o problemă în alte probleme mai mici și mai ușor de gestionat, precum:

- Cum introducem și cum scoatem rufele din mașina de spălat?
- Cum anume intră și iese apa din mașina de spălat?
- Cum anume ne asigurăm că apa este la temperatura adecvată?

La rândul lor, aceste mici probleme pot fi descompuse în probleme și mai mici:

- Care este poziția optimă a ușii în cadrul mașinii de spălat?
- Cum putem face ca ușa să se închidă etanș, astfel încât apa să nu se scurgă?



Descompunerea unei probleme legate de proiectarea unei mașini de spălat rufe

Algoritmi

Algoritmii pot fi elaborați pentru a specifica pașii exacți pe care mașina ar trebui să îi urmeze pentru diferite cicluri de spălare. În plus, sunt necesari mult mai mulți algoritmi. Spre exemplu, poate fi elaborat un algoritm care să specifice secvența exactă de pași ce trebuie urmați pentru fabricarea unei mașini de spălat rufe.

Evaluare

Proiectarea unui produs reprezintă o provocare majoră și ca urmare, proiectele inițiale vor fi evaluate continuu. Prin intermediul acestei evaluări, proiectanții determină unde și cum poate aduce îmbunătățiri produsului. Spre exemplu, dacă în timpul testării unei mașini de spălat rufe, apa se scurge în exteriorul acesteia, designul produsului poate fi modificat astfel încât să se prevină acest lucru în viitor.

Generalizare

Aspectul unei mașini de spălat poate fi de asemenea îmbunătățit făcându-l universal, prin utilizarea pe butoane a simbolurilor în locul cuvintelor, astfel încât mașina de spălat să poată fi utilizată de orice persoană, indiferent de limbă. O altă metodă ar fi prin proiectarea unei mașini de spălat care să funcționeze atât la 110V, cât și la 220V pentru a putea fi folosită în diferite țări. Dacă designul mașinii de spălat este modificat în acest fel, poate fi generalizat.

Recunoașterea de șabloane

Recunoașterea de șabloane are un rol important și, împreună cu celelalte cinci tehnici, poate fi utilizată de mai multe ori în diferite etape ale procesului de proiectare. Poate ajuta în procesul de abstractizare prin evidențierea similarităților și diferențelor în aspectele diferite ale unei probleme. Recunoașterea de șabloane poate ajuta și procesul de generalizare, unde anumite aspecte ale unei probleme specifice pot fi relaționate cu probleme mai generale. În mod similar, evaluarea aspectului ajută în procesul de generalizare prin evaluarea modului în care acesta poate fi aplicat pentru diferite setări.

Cele 6 tehnici de gândire computațională nu trebuie aplicate obligatoriu într-o anumită ordine sau în cadrul unei singure etape a procesului de proiectare. Ele pot fi utilizate în diverse etape de proiectare și într-o ordine diferită.

Pași

Exemplu: Organizarea unui festival de muzică

În acest exemplu, tehnicile de gândire computațională sunt aplicate în cadrul unei probleme complexe precum organizarea unui festival de muzică.

Abstractizarea

Pentru a organiza un festival de muzică, trebuie să înțelegeți în primul rând ce anume este un festival de muzică. Ca urmare, primul pas ar putea consta în enumerarea elementelor esențiale pe care un festival de muzică ar trebui să le conțină:

- Muzicieni
- Locație
- Marketing și publicitate
- Bilete
- Personal angajat etc.

Există și alte detalii care trebuie luate în considerare la organizarea unui festival, dar care nu sunt esențiale, precum culoarea biletelor sau formularea exactă de pe bilete. Acestea pot îngreuna planificarea inițială și pot fi elaborate ulterior. Utilizând **abstractizarea**, detaliile inutile pot fi eliminate din plan, pentru moment.

Descompunere și Algoritmi

Provocările majore care apar în organizarea unui festival de muzică pot fi divizate în sarcini mai mici. În practică, acestea pot fi delegate spre a fi rezolvate diverselor persoane:

- O persoană care să se ocupe de rezervarea locației de desfășurare a festivalului.
- O persoană responsabilă de marketing și publicitate.
- O persoană responsabilă cu gestionarea rezervărilor și biletelor.

Aceste sarcini mai mici pot fi la rândul lor divizate în probleme și mai mici. Acest lucru reprezintă **descompunerea** unei probleme și este un exemplu de aplicare a metodelor de gândire computațională.

De exemplu, problema organizării locației poate fi divizată în sarcini mai mici, precum:

- Unde este situată locația?
- Când este locația disponibilă?
- Există parcare disponibilă?

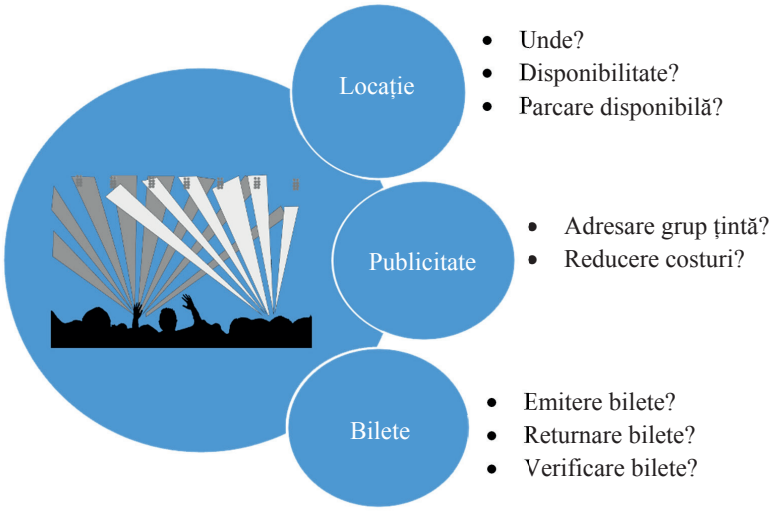
Similar, sarcina legată de parcarea mașinilor poate fi **descompusă** în probleme și mai mici. De asemenea, pot fi creați **algoritmi** simpli, cum ar fi:

- Pentru a direcționa mașinile spre locurile de parcare destinate.
- Pentru a direcționa mașinile spre parcul auto atunci când parcarea principală este plină.
- Pentru a preveni blocarea rutelor importante.
- Pentru a asigura faptul că parcarea prioritară este în permanență disponibilă pentru persoanele VIP.

Problema publicității poate fi descompusă după cum urmează:

- Cum putem prezenta festivalul astfel încât persoanele să fie interesate să participe?
- Cum să se realizeze promovarea evenimentului către publicul țintă într-un mod eficient și rentabil?

Și procesul de rezervare și vânzare de bilete poate fi de asemenea **descompus**. Spre exemplu, procesul de vânzare de bilete poate fi divizat într-un set de pași și se poate crea un **algoritm** ce acoperă proiectarea, tipărirea, cumpărarea, livrarea, și returnarea biletelor.



Descompunerea unei probleme legate de organizarea unui festival de muzică

Evaluare

După terminarea evenimentului, poate fi utilă colectarea de feedback cu privire la ce a mers bine și ce nu. Aceasta este o metodă de evaluare a planului. Dacă festivalul se mai organizează și anul viitor, lecțiile învățate din evaluarea realizată anul acesta vor îmbunătăți calitatea evenimentului de anul viitor.

Generalizare

Generalizarea este de asemenea relevantă. O echipă care a gestionat cu succes un festival de muzică își poate stabili obiective mai ambițioase, spre exemplu un turneu de festivaluri sau creșterea duratei festivalului, adăugarea mai multor acte artistice sau a mai multor genuri de muzică. Soluția existentă este reutilizată, însă poate fi generalizată pentru a include elemente noi.

1.2 INSTRUCȚIUNI

Concepte

Gândirea computațională reprezintă o abordare generală de rezolvare a problemelor, însă poate fi de asemenea folosită ca punct de plecare pentru crearea instrucțiunilor pentru computere.

Descompunerea problemelor în pași mai simpli a condus la dezvoltarea unuia sau mai multor **algoritmi** - colecții de pași simpli, bine definiți, ce trebuie urmați pentru a rezolva o problemă. Algoritmii pot fi apoi prezentați într-o formă pe care computerele o pot înțelege.

Pași

Exemplu: Un algoritm de sortare a oamenilor în funcție de înălțime

În acest exemplu, trebuie sortați elevii dintr-o clasă în funcție de înălțime. Pentru aceasta, vați putea decide asupra unui număr de pași de urmat:

Pas 1: Aliniați toate persoanele pe un singur rând.

Pas 2: Decideți în ce capăt al rândului vor fi persoanele 'înalte' și în ce capăt cele 'scunde'.

Pas 3: Comparați în mod repetat înălțimile elevilor și schimbați locurile elevilor atunci când ei se află într-o poziție greșită.

Acest set de pași reprezintă un algoritm. Totuși, nu este un algoritm foarte detaliat. Pasul 3, spre exemplu, ar putea fi descompus la rândul lui în mai multe probleme mai mici:

- Cu ce capăt al rândului se începe
- Cum se compară înălțimea elevilor
- Cum se procedează dacă elevii sunt mai înalți sau mai scunzi

Odată ce este construit în mod corect, acest algoritm de sortare a persoanelor după înălțime ar putea fi modificat astfel încât să rezolve diferite alte probleme. O variantă a acestui algoritm ar putea sorta elevii dintr-o clasă în funcție de data nașterii. Și, dacă se adaugă câteva modificări, algoritmul ar putea fi utilizat pentru a determina persoanele născute în aceeași zi.

Concepte

Algoritmi pentru computere

În algoritmul de sortare a elevilor după înălțime, instrucțiunile erau suficient de detaliate pentru a putea fi înțelese și executate de o persoană, însă nu erau suficient de detaliate astfel încât un computer să le poată executa. Un computer trebuie programat utilizând un limbaj de programare, care este mult mai detaliat și mai precis decât un limbaj uman.

Ca urmare, pentru ca un computer să execute instrucțiunile din algoritm, acesta trebuie convertit într-un limbaj pe care computerul să îl poată înțelege. Un algoritm exprimat într-o formă ce poate fi înțeleasă și executată de către un computer poartă numele de **program**.

Atunci când computerul urmează instrucțiunile descrise în program, se spune că el rulează sau execută programul respectiv. În concluzie, un program este scris într-un limbaj de programare, iar computerul rulează sau execută programul.

Imaginați-vă un robot care coace o prăjitură. O instrucțiune dintr-un algoritm de a 'introduce prăjitura în cuptor' nu ar fi suficient de detaliată astfel încât robotul (computerul) să o execute. Un robot necesită un program conținând mult mai multe instrucțiuni detaliate despre cum anume să își miște mâinile și degetele atunci când pune prăjitura în cuptor și despre cum să nu o răstoarne.

1.3 EXERCIȚII RECAPITULATIVE

1. Computing reprezintă un set de activități ce include:
   a. efectuarea unor calcule sau procesarea unor date.
   b. respectarea unei rețete pentru coacerea biscuiților.
   c. observarea corectă și atentă a stelelor, pentru o perioadă lungă de timp.
   d. observarea corectă și atentă a reacțiilor chimice.

2. Gândirea computațională reprezintă:
   a. rezolvarea unei probleme dificile de către un computer.
   b. procesul de analiză a problemelor și provocărilor și de identificare a soluțiilor posibile pentru rezolvarea acestora.
   c. utilizarea unui computer pentru efectuarea multor calcule matematice.
   d. orice activitate în care o persoană lucrează la un computer.

3. Care dintre următoarele NU reprezintă o metodă de gândire computațională?
   a. Abstractizarea
   b. Înțelegerea
   c. Descompunerea
   d. Recunoașterea de șabloane

4. Care dintre următoarele reprezintă un bun exemplu de descompunere a unei probleme?
   a. Divizarea sarcinii de a proiecta un robot în sarcini mai mici (proiectarea mâinii, proiectarea sursei de alimentare, proiectarea senzorilor).
   b. Tăierea unei prăjituri în 6 felii egale.
   c. Plasarea unei măr într-un borcan de sticlă și fotografierea lui zilnică pentru a monitoriza degradarea acestuia.
   d. Utilizarea unui motor de căutare pentru a afla răspunsul la o întrebare.

5. Un program reprezintă:
   a. Regulile detaliate ale unui joc sau unui sport.
   b. Un algoritm exprimat într-o formă adecvată pentru un computer.
   c. O colecție de legi și reglementări care determină ce clădiri pot fi construite legal într-un anumit loc.
   a. O secvență de instrucțiuni ce trebuie urmate de o persoană (de exemplu o rețetă de prăjitură).

6. Care dintre acestea este cel mai puțin probabil să reprezinte o modalitate prin care algoritmii sunt utilizați în gândirea computațională?
    a. Un algoritm poate duce la un program de calculator care, atunci când este rulat, rezolvă problema.
    b. Un algoritm poate duce la un program de calculator care folosește intuiția pentru a obține soluții mai bune.
    c. Un algoritm poate furniza instrucțiuni pas cu pas pentru fabricarea unui obiect.
    d. Un algoritm poate furniza instrucțiuni pas cu pas pentru diverse procese ce implică oameni, bani și alte resurse.

7. Mai jos sunt explicate 3 rețete:

    Tort de ciocolată:
    Setați cuptorul la 180ºC
    Ungeți cu unt 2 tăvi de tort de 9" fiecare
    Amestecați ingredientele cu un tel timp de 1 minut
    Puneți amestecul obținut în tăvile de tort
    Coaceți timp de 30 de minute
    Lăsați torturile să se răcească
    Ornați torturile și apoi lăsați-le la rece câteva ore.

    Turtă dulce:
    Amestecați ingredientele până obțineți un aluat moale
    Setați cuptorul la 190ºC
    Întindeți aluatul cu sucitorul într-o foaie groasă de 1/8" și apoi tăiați-l în diverse forme
    Coaceți până când marginile sunt tari, aproximativ 10 minute.

    Brioșe cu afine:
    Setați cuptorul la 185ºC
    Ungeți cu unt sau ulei 18 forme de brioșe
    Bateți untul și zahărul până se obține o cremă consistentă
    Adăugați celelalte ingrediente
    Turnați compoziția în formele de brioșe
    Glazurați-le cu topping de ciocolată
    Coaceți timp de 15 - 20 minute.

    Care dintre următoarele reprezintă un șablon comun pentru toate cele 3 rețete?
    a. Ungeți cu unt sau ulei 18 forme de brioșe.
    b. Ornați și apoi lăsați la rece.
    c. Întindeți aluatul.
    d. Setați cuptorul la o anumită temperatură.

8. Care dintre următoarele ar reprezenta cel mai relevant detaliu în proiectarea unui program de gătit pentru un robot?
    a. Culoarea robotului.
    b. Ce cantități să utilizeze din fiecare ingredient.
    c. Din ce magazin au fost cumpărate ingredientele.
    d. Dacă programatorul este dreptaci sau stângaci.

CAPITOLUL 2 –
DEZVOLTARE SOFTWARE

La finalul acestui capitol, veți putea să:

- Înțelegeți diferența dintre un limbaj formal și un limbaj natural
- Definiți termenul de cod și să înțelegeți distincția dintre codul sursă și codul mașină
- Înțelegeți termenii de descriere a unui program și specificații
- Recunoașteți etapele de creare a unui program: analiză, proiectare, programare, testare, îmbunătățire.

2.1 PRECIZIA LIMBAJULUI

Concepte

Tipuri de limbaje:

- **Limbaj natural**

Limbile vorbite precum Engleza, Franceza sau Chineza reprezintă limbi naturale. O limbă naturală are nevoie de un anumit context pentru a fi clar înțeleasă și pentru a se evita ambiguitățile.

- **Limbaj formal**

Un limbaj formal este foarte strict structurat, cu reguli exacte și precise. Este utilizat în matematică, ecuații chimice și programe de calculator. Este clar și lipsit de ambiguitate.

Limbajele de programare reprezintă limbaje formale. O modalitate prin care limbajele formale pot evita ambiguitățile constă în utilizarea parantezelor pentru a grupa cuvinte și termeni și prin evitarea cuvintelor precum 'el' sau 'ea' într-un context unde adresarea nu este foarte clară

Regăsiți mai jos 3 tipuri diferite de paranteze utilizate în cadrul limbajelor formale. Fiecare tip de paranteză are o funcție diferită și poate fi utilizată pentru a defini diverse lucruri, cum ar fi ordinea instrucțiunilor pe care trebuie să le execute un program.

PARANTEZĂ	NUME
( )	Paranteze rotunde.
{ }	Acolade.
[ ]	Paranteze drepte.

Pași

Exemplu: Ambiguitate cu 'și și 'sau'

Dacă întrebi o persoană ce aromă de bomboane preferă, aceasta poate răspunde: "Mure și Lămâie sau Zmeură și Portocală". Vei înțelege răspunsul, însă pentru ca un computer să poată înțelege această propoziție, cuvintele trebuie grupate în mod corect.

Pentru un computer, trebuie să scrii (Mure ȘI Lămâie) SAU (Zmeură ȘI Portocală)

(Mure ȘI Lămâie) SAU (Zmeură ȘI Portocală)

În timp ce, dacă scrii Mure ȘI (Portocală SAU Lămâie) ȘI Zmeură, computerul inerpretează astfel:

Mure ȘI (Portocală SAU Lămâie) ȘI Zmeură

2.2 LIMBAJE DE PROGRAMARE

Concepte

Limbajele de programare sunt proiectate pentru a scrie programe care instruiesc computerele să execute o secvență de instrucțiuni în scopul rezolvării unei probleme. Limbajele de programare au un vocabular mai limitat decât limbajele naturale.

Există diverise limbaje de programare. Acest material de instruire face referire la un cunoscut limbaj de programare numit Python. Alte limbaje de programare foarte cunoscute sunt Java și C++.

Concepte

Coding

Textul dintr-un program, scris sub forma unei secvențe de instrucțiuni pe care computerul să le execute, poartă numele de cod. Diversele limbaje de programare utilizează diferite stiluri de cod, cu reguli diferite și modalități diferite de organizare a instrucțiunilor, cunoscute sub denumirea de sintaxă.

Scrierea unui program poartă numele de programare sau coding.

Persoanele care scriu programe se numesc programatori.

Există 2 tipuri de cod: **cod sursă** și **cod mașină**.

- **Codul sursă** reprezintă codul scris de programator pe care oamenii îl pot înțelege. Acesta este introdus în computer, de obicei sub formă de text, semne de punctuație și simboluri și conține instrucțiunile pentru computer. Dacă înveți limbajul de programare formal și regulile lui (e.g. Python), poți scrie cu succes cod sursă.

- **Codul mașină** reprezintă o serie de 1 si 0, creată de computer pe baza codului sursă, pe care circuitele electronice ale computerului le pot înțelege. Crearea codului mașină pe baza codului sursă este cunoscută sub denumirea de compilare sau interpretare a codului sursă.

Pași

Exemplu: Cod mașină pentru blocarea unei uși

Instrucțiunea de blocare a ușii scrisă în cod sursă ar putea arăta astfel:

Lock(door)

Chiar dacă nu este corect din punct de vedere gramatical, are sens pentru oamenii care îl citesc.

Un computer care citește instrucțiunile codului sursă vede literele 'L', 'o', 'c', 'k', '(', și așa mai departe. Instrucțiunile nu sunt într-un format pe baza căruia computerul să poată executa o acțiune. În schimb, computerul are nevoie ca aceste instrucțiuni să fie convertite într-un format pe care el îl poate înțelege și executa.

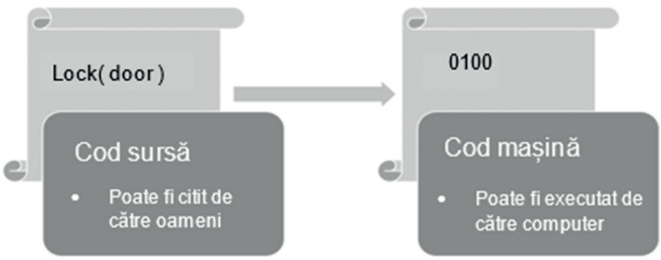
Computerul conține circuite electrice care funcționează conform unor anumite șabloane sau combinații de 1 și 0. Mai jos, regăsiți un exemplu despre ce ar putea însemna anumite șabloane de 1 și 0.

Cod mașină	Sens al instrucțiunii codului mașină
0001	Pornește alarma
0010	Blochează ferestrele
0100	Blochează ușa
0110	Blochează ferestrele și ușa
1000	Pornește sistemul de aspersoare
1001	Pornește sistemul de aspersoare și alarma

Instrucțiunile precum '0110' reprezintă instrucțiuni cod mașină.

În realitate, gama completă de posibile instrucțiuni cod mașină pentru un computer poate fi mult mai largă. Ar fi, de exemplu, multe instrucțiuni pentru efectuarea calculelor aritmetice.

Codul sursă este convertit în cod mașină înainte ca un computer să execute instrucțiunile. 'Lock(door)' ar fi tradus în 0100 și atunci computerul ar putea executa instrucțiunea.



Cod sursă vs Cod mașină

În acest exemplu simplu, o instrucțiune cod sursă 'Lock(door)' corespunde unei instrucțiuni cod mașină, 0100. În mod normal, o singură linie de cod sursă necesită mai multe instrucțiuni cod mașină scrise una după cealaltă, în ordinea corectă.

În trecut, programatorii converteau codul sursă în cod mașină manual și creau și documentau singuri seriile de 1 și 0. Apariția programelor care traduc codul sursă în cod mașină a permis scrierea unor programe mult mai mari și mai complexe.

2.3 TEXT ÎN CADRUL CODULUI

Concepte

Pe lângă scrierea codului sursă, programatorii trebuie să își documenteze munca sub forma unor notițe de tip text. Aceste notițe nu sunt citite de către computer, însă îi ajută pe programatori și pe ceilalți colegi să înțeleagă diferitele etape de dezvoltare ale unui program.

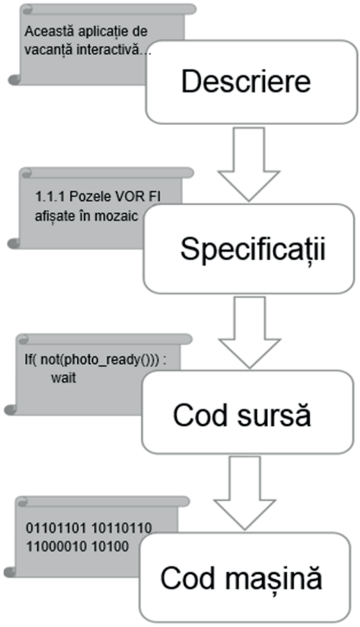
Text ce descrie un program

Programatorii scriu descrierea și specificațiile unui program pentru a ajuta persoanele implicate în proiect să înțeleagă care este scopul programului și ce problemă trebuie acesta să rezolve. Acestea pot fi de asemenea utilizate și în etapa de evaluare pentru a verifica dacă programul funcționează așa cum ar trebui.

O **descriere a unui program** explică scopul acestuia și modul său de funcționare. Acest lucru este util și pentru alți programatori, pentru utilizatorii produsului final și pentru departamentul de marketing în scopuri de promovare și vânzare.

O **specificație a unui program** reprezintă un set de cerințe care evidențiază ce anume va face programul. În mod normal, specificațiile sunt scrise înaintea programului. O specificație este mult mai detaliată decât o descriere și prevede cerințe ce pot fi testate.

Spre exemplu, o specificație poate include cerința "Programul VA ÎNCHIDE monitorul computerului dacă calculatorul nu a fost utilizat timp de 1 minut, pentru a economisi energia electrică". Odată ce programul este scris, el poate fi testat lăsând computerul să funcționeze timp de 1 minut și verificând dacă într-adevăr programul închide monitorul. Imaginea de mai jos descrie etapele posibile de realizare a unui program, de la proiectare la execuție.



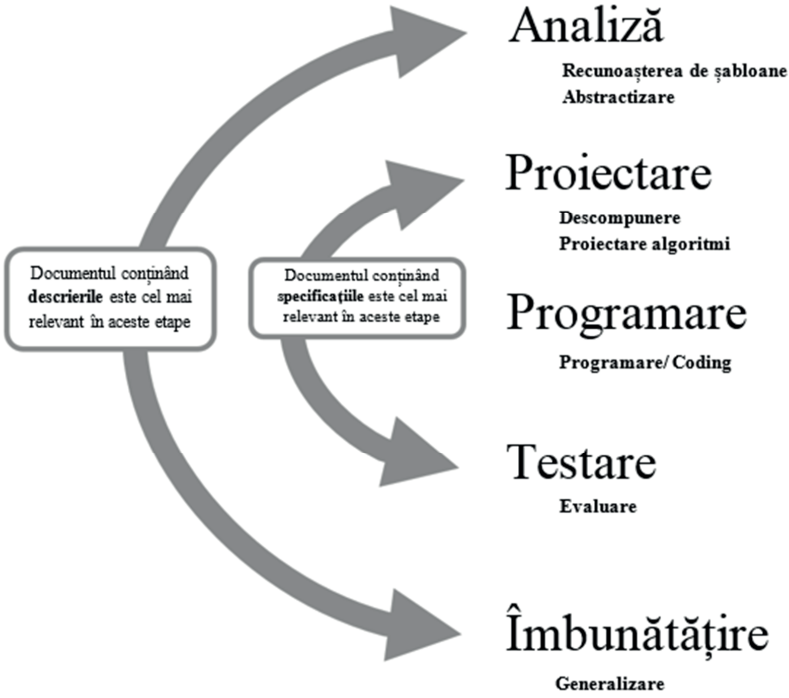
Etapele de creare a unui program

2.4 ETAPELE DE DEZVOLTARE ALE UNUI PROGRAM

Concepte

Descrierile şi specificaţiile pot juca un rol important în dezvoltarea unui program.

Următoarea diagramă ilustrează câteva dintre etapele şi activităţile principale în crearea unui program.



Etapele dezvoltării unui Program

Analiză

Această etapă implică definirea problemelor care trebuie rezolvate. Analiza reprezintă, în linii mari, un proces de abstractizare, prin care se enumeră toate aspectele problemei, identificându-se aspectele relevante şi modul în care sunt acestea interconectate.

Proiectare

Această etapă implică lucrul cu algoritmi (seturi de paşi de urmat) în scopul rezolvării problemei. Astfel, se utilizează descompunerea pentru a diviza problema în părţi mai mici şi se planifică modul de proiectare al algoritmilor.

Programare

Această etapă presupune scrierea programului. Acest lucru implică identificarea unei modalităţi pentru a exprima algoritmii în limbajul de programare ales.

Testare

Această etapă presupune verificarea faptului că programul funcționează așa cum ar trebui. În etapa de testare se pot identifica eventuale erori logice sau de sintaxă ale programului. Acest aspect este acoperit în detaliu în capitolul 15.

Îmbunătățire

Această etapă implică adăugarea de noi caracterisitici pentru a extinde funcționalitățile programului, pentru îmbunătățirea performanțelor acestuia sau pentru a generaliza programul în scopul utilizării sale în diverse situații.

Descrierea generală a programului este de obicei formulată foarte devreme în cadrul proiectului, ca parte a stabilirii scopului programului.

Specificațiile programului sunt elaborate în mod normal în etapa de proiectare. O parte dintre cerințele din specificații pot apărea direct din descompunerea problemei în componente mai mici. Specificațiile programului sunt examinate în etapa de testare întrucât fiecare afirmație legată de funcționalitățile programului trebuie verificată.

În etapa de îmbunătățire, descrierea programului poate fi actualizată pentru a include planuri de extindere a funcționalităților oferite de program.

2.5 EXERCIȚII RECAPITULATIVE

1. Un limbaj formal este:
   a. Un limbaj în care nu este posibil să faci greșeli.
   b. Un limbaj cu reguli clar definite și înțelesuri precise.
   c. Cel mai bun limbaj ce poate fi folosit la scrierea unei vederi.
   d. Orice limbaj care conține cuvintele "și" și "sau".

2. Engleza, Araba și Chineza sunt:
   a. Limbaje formale.
   b. Limbaje de programare.
   c. Limbaje naturale.
   d. Cod sursă.

3. Un cod mașină este:
   a. Tradus în cod sursă astfel încât computerul să poată executa instrucțiunile.
   b. O metodă de comunicare în siguranță între 2 computere.
   c. Un șir de 1 și 0 pe care computerul îl execută.
   d. Un acord scris între un programator și un client final în care se specifică funcționalitățile programului.

4. Un program scris în limbajul de programare Python este un exemplu de:
   a. Cod sursă
   b. Cod mașină
   c. O specificație a programului
   d. O descriere a programului

5. O specificație a programului reprezintă:
   a. Codul care va fi executat de computer.
   b. Comentariile din cadrul codului pe care un programator le citește.
   c. Semnale electrice din computer, emise atunci când un program este rulat.
   d. O descriere a funcționalităților programului, utilizată în perioada de proiectare a acestuia.

6. Potriviți fiecare etapă din crearea unui program (de la a la e) cu scopul ei:
   a) Testare, b) Proiectare, c) Programare, d) Analiză, e) Îmbunătățire

Îmbunătățirea programului existent.	
Definirea clară a problemei.	
Verificarea funcționării corecte a programului.	
Proiectarea algoritmilor într-un limbaj de programare ales.	
Elaborarea unor algoritmi pentru rezolvarea problemei.	

CAPITOLUL 3 –
ALGORITMI

La finalul acestui capitol, veți putea să:

- Definiți termenul de secvență de instrucțiuni și să identificați scopul utilizării secvențelor de instrucțiuni în crearea algoritmilor.
- Recunoașteți metodele posibile de reprezentare a problemelor: scheme logice, pseudocod
- Recunoașteți simbolurile din schemele logice, precum: start/stop, proces, decizie, intrare/ieșire, conector, săgeată
- Identificați secvențe de operații reprezentate în cadrul unei scheme logice sau unui pseudocod.
- Scrieți un algoritm corect pe baza unei descrieri, utilizând o schemă logică sau un pseudocod
- Remediați erorile dintr-un algoritm precum: elemente de program lipsă, secvență incorectă de instrucțiuni, rezultat incorect

3.1 PAȘII UNUI ALGORITM

Concepte

Secvențe de instrucțiuni

Algoritmul este o succesiune finită de paşi care rezolvă o anumită cerință/problemă. Altfel spus, algoritmii reprezintă probleme complexe descompuse în paşi sau **instrucțiuni** mai simple. În majoritatea algoritmilor, instrucțiunile sunt executate una după cealaltă, într-o anumită ordine (**secvență**). O secvență reprezintă un număr de instrucțiuni simple ce ar fi trebui executate una după cealaltă.

În majoritatea algoritmilor, contează ordinea de executare a instrucțiunilor. Un robot trebuie să pună amestecul de ingrediente în tavă înainte de a introduce tava în cuptor.

Proiectarea secvenței de instrucțiuni reprezintă o competență crucială în programare. Un programator trebuie să se asigure că toate acțiunile cerute sunt executate în ordinea corectă pentru a îndeplini sarcina sau setul de sarcini. O secvență de instrucțiuni este metoda fundamentală de control în cadrul unui program. Secvența de instrucțiuni este decisă în etapa de proiectare a programului, unde se utilizează algoritmi și scheme logice pentru a crea cea mai eficientă și corectă secvență de control a programului.

Intrare și ieșire

Instrucțiunile utilizează de cele mai multe ori informații din lumea exterioară computerului, pe care apoi le prelucrează într-un anumit fel. Programele primesc informațiile de la utilizator prin intermediul datelor de intrare (**input**) și generează un rezultat, anumite date de ieșire (**output**).

Date de intrare (input)

Reprezintă o valoare aflată în afara computerului, necesară pentru executarea instrucțiunilor. De exemplu: o măsurare a temperaturii sau un număr tastat pe tastatura unui sistem de securitate.

Date de ieșire (output)

Reprezintă o valoare calculată sau o acțiune efectuată de către program, afișată către lumea exterioară. De exemplu: un led care se aprinde și se stinge, o alarmă care este setată sau un mesaj afișat pe ecran.

Pași

Exemplu: Intrări și ieșiri pe un telefon mobil.

Un telefon mobil preia informații (input), le procesează și returnează rezultate (outputs).

- Touch screen-ul este o metodă importantă de introducere a datelor, de exemplu când scrieți un mesaj sau când apăsați pe ecran pentru a deschide o aplicație.
- Ecranul de afișare reprezintă o metodă importantă de afișare a unor date, ca de exemplu afișarea unei liste de contacte.

Decizii de tip Da/Nu

Așa cum ați văzut în capitolele anterioare, ordinea instrucțiunilor este importantă în programare. În general, instrucțiunile sunt executate unele după celelalte, dar uneori trebuie luate decizii cu privire la acțiunile viitoare. Algoritmul poate fi proiectat pentru a pune întrebări utilizatorului și a aștepta un răspuns din partea acestuia pentru a decide următorul pas.

O **decizie de tip da / nu (yes/no)** reprezintă o instrucțiune care alege următoarea instrucțiune de executat. Acest lucru se realizează în funcție de răspunsul pozitiv sau negativ introdus de utilizator.

Pași

Exemplu: Așteptarea ca un cuptor să se încălzească

Înainte de a adăuga amestecul de ingrediente în cuptor, cuptorul trebuie să fie încălzit la temperatura potrivită. Un algoritm ar putea include o decizie da/nu pentru a determina acest lucru. "Cuptorul este la temperatura potrivită?" Un termostat va furniza răspunsul (da sau nu). Dacă răspunsul este da, atunci prăjitura poate fi introdusă la cuptor. Dacă răspunsul este nu, algoritmul va mai aștepta o perioadă și va pune din nou întrebarea.

3.2 METODE DE REPREZENTARE A ALGORITMILOR

Concepte

2 tehnici care îi ajută pe programatori în descrierea algoritmilor sunt pseudocodul și schemele logice (flowchart). Acestea sunt utilizate pentru reprezentarea secvenței de pași din cadrul unui algoritm. Algoritmii exprimați în pseudocod sau sub forma schemelor logice sunt rareori suficient de precișI pentru a putea fi utilizați de un computer, dar prezintă o serie de pași suficient de în detaliu astfel încât un programator să poată înțelege modul în care ar trebui să funcționeze algoritmul respectiv.

Pseudocod:

Un pseudocod reprezintă o modalitate informală de reprezentare a unui algoritm, utilizând instrucțiuni scrise într-un limbaj natural, ca de exemplu Engleza. Pașii din cadrul algoritmului sunt scriși sub forma unei secvențe de instrucțiuni. Un algoritm scris ca pseudocod este destinat mai curând să fie citit de oameni decât de un computer.

Pseudocodul arată ca un cod în care programul va fi scris, însă nu este atât de precis. Dacă pseudocodul poate fi înțeles, atunci următorul pas este scrierea efectivă a codului din program.

Un algoritm pentru perierea dinților, scris în pseudocod, ar putea arăta așa:

Puneți pastă pe periuță
Periați dinții din dreapta sus
Periați dinții din dreapta jos
Periați dinții din stânga sus
Periați dinții din stânga jos
STOP

Schemă logică (flowchart)

O schemă logică reprezintă o modalitate grafică de reprezentare a unui algoritm. Schemele logice ilustrează o serie de paşi simpli folosind săgeţi care să indice progresul de la un pas la altul. Paşii sunt reprezentaţi utilizând blocuri de text de diverse forme. O schemă logică reprezintă un instrument fundamental utilizat în dezvoltarea unui program. Ea permite programatorilor să evidenţieze, pas cu pas, cum doresc ei să rezolve o problemă sau cum şi-ar dori să funcţioneze un program.

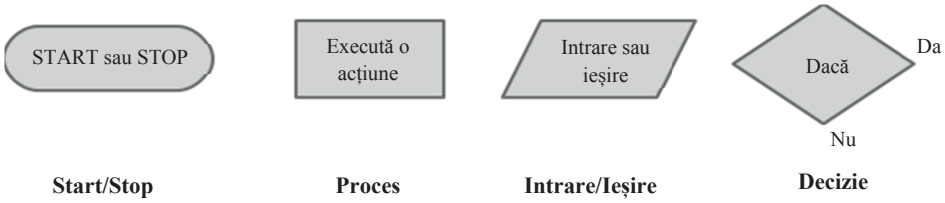
Algoritmul pentru perierea dinţilor ar putea fi reprezentat şi sub forma unei scheme logice:

Schemă logică pentru perierea dinţilor

3.3 SCHEME LOGICE

Concepte

Mai jos sunt prezentate câteva forme de blocuri utilizate în cadrul unei scheme logice:



Simbolurile unei scheme logice

Blocurile reprezintă următoarele:

Start sau Stop

Algoritmul pornește de la caseta Start și rulează până ajunge la caseta Stop.

Proces

Aceste blocuri conțin instrucțiuni simple pentru a executa o acțiune, cum ar fi adunarea a 2 numere sau căutarea unui cuvânt în dicționar.

Intrare sau ieșire

Aceste blocuri sunt destinate interacțiunii cu lumea exterioară computerului. Spre exemplu, o intrare ar putea fi de la un senzor de temperatură. O ieșire ar putea fi o acțiune precum aprinderea sau stingerea unei lumini.

Decizie

Blocurile de decizie permit alegeri alternative privind pașii următori pe care algoritmul trebuie să îi execute. Ele de obicei conțin o întrebare cu 2 variante de răspuns: da și nu. Dacă răspunsul este 'da' se urmează un anumit scenariu. Dacă răspunsul este 'nu', se urmează alt scenariu. Aceasta este metoda prin care algoritmii pot merge dincolo de secvențele simple de pași.

Conectarea blocurilor

Există încă 2 simboluri adiționale în cadrul unei scheme logice:

Săgețile și conectorii sunt folosiți pentru a conecta blocurile din cadrul unei scheme logice.

Săgeată

O linie direcțională desenată pentru a conecta 2 blocuri în cadrul unei scheme logice. Această săgeată indică direcția de urmat în cadrul unui algoritm. De obicei, acestea sunt

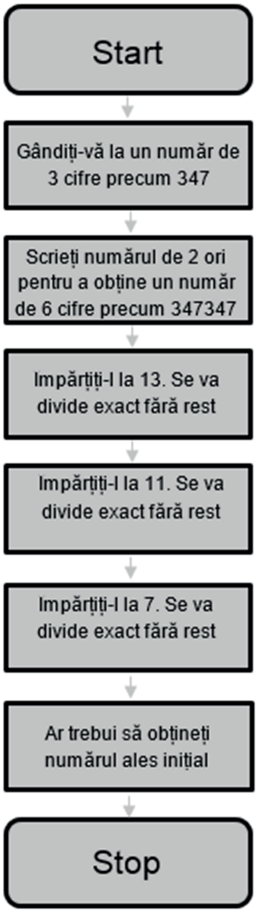
denumite linii de flux. De exemplu, instrucțiunile din cadrul unei secvențe conțin săgeți între ele.

Conector

Un mic cerc în cadrul unei scheme logice este utilizat pentru a conecta două linii de flux între ele. El indică un salt de la un punct din proces la altul, ca de exemplu atunci când se răspunde la o întrebare "Da/Nu".

Pași

Exemplu: Schemă logică pentru 'Trucul Magic 347347'



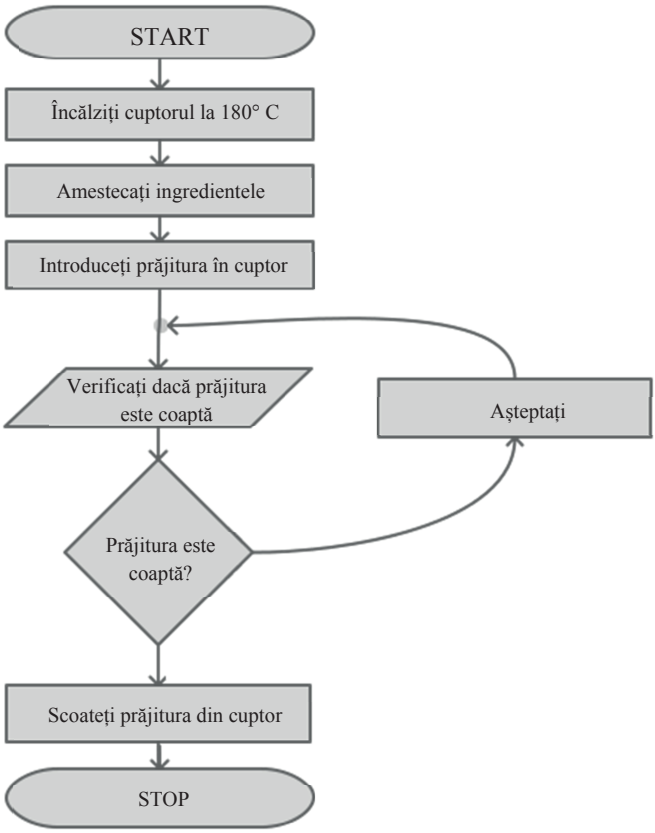
Schemă logică pentru un truc numeric

Aceasta este schema logică pentru un truc magic cu numere. Ea ilustrează o simplă secvență de instrucțiuni, fără blocuri de decizie. Se pornește procesul de la blocul Start și se continuă urmând liniile de flux, executând instrucțiunile din fiecare bloc, până se ajunge la blocul Stop.

Exemplu: Schemă logică pentru coacerea unei prăjituri

Această schemă logică reprezintă un algoritm de coacere a unei prăjituri. Ea conține un bloc de decizie. Acesta determină dacă prăjitura poate fi scoasă din cuptor sau, dacă nu, îl instruiește pe bucătar să aștepte până când prăjitura este gata și poate fi scoasă din cuptor.

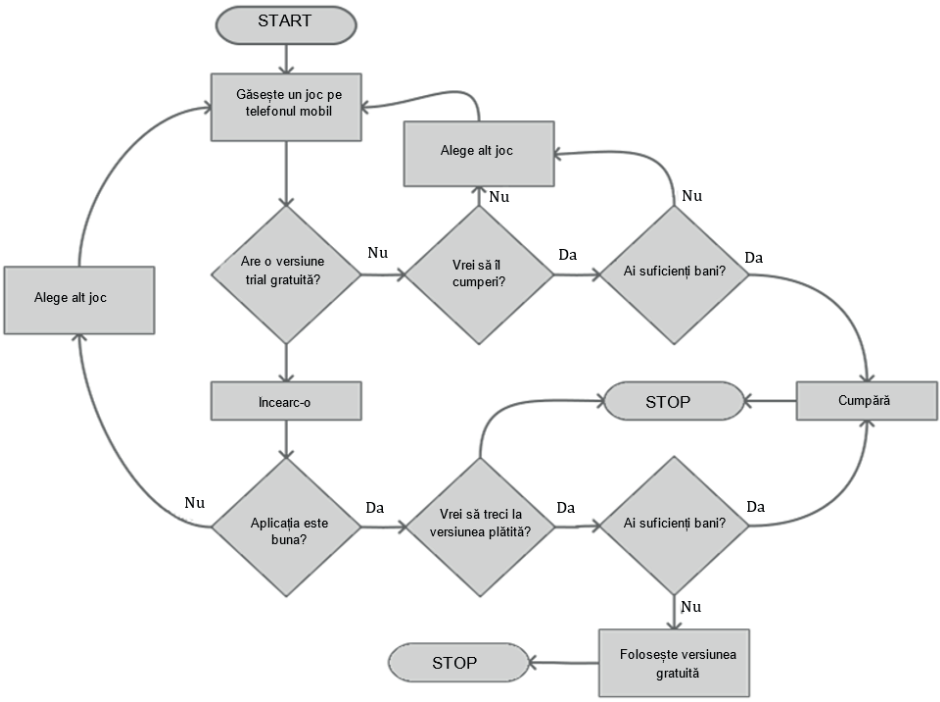
"Prăjitura este coaptă?" reprezintă o dată de intrare în cadrul algoritmului și este reprezentată printr-un bloc de intrare/ieșire.



Schema logică pentru coacerea unei prăjituri

Exemplu: Schemă logică pentru alegerea unui joc pe telefonul mobil

Această schemă logică arată secvența de acțiuni pentru a alege un joc pe telefonul mobil. Fiecare bloc în formă de romb conține câte o întrebare. Răspunsul la întrebare poate fi da sau nu. Răspunsul determină ce bloc urmează.



Schemă logică pentru alegerea unui joc pe telefonul mobil

Exemplu: Schemă logică pentru trezirea dimineața

Creați o schemă logică pentru a vă trezi dimineața, a lua micul dejun și a pleca la școală.

INDICII:

1. Porniți cu o secvență de acțiuni și scrieți acei pași ca pseucocod, sub forma unei liste numerotate.
2. Luați anumiți pași și descompuneți-I în pași mai mici. De exemplu, dacă aveți un pas numit 'pregătește micul dejun' – acesta poate fi divizat în pași mai mici precum "pune pâinea în prăjitor".
3. Reprezentați această secvență mai detaliată sub forma unei scheme logice.
4. Faceți schema logică mai interesantă prin introducerea variațiilor. De exemplu, cum afectează vremea drumul către școală? Variațiile pot depinde de întrebări precum:
   - Plouă?
   - Întârzii la școală?
5. Adăugați blocuri de decizie pentru fiecare dintre aceste întrebări, cu diferiți pași dacă răspunsul este da sau nu.

3.4 PSEUDOCOD

Concepte

În loc să creați o schemă logică, un algoritm poate fi reprezentat în pseudocod.

Pseudocod reprezintă o modalitate de a scrie în mod informal modul de operare al unui algoritm. El combină limbajul natural informal (e.g. Engleza) cu porțiuni din structura limbajelor de programare.

Pași

Exemplu: Pseudocod pentru 'Trucul Magic' 347347

Schema logică pentru trucul numeric magic ar putea fi scrisă sub formă de pseudocod astfel:

Gândiți-vă la un număr de 3 cifre precum 347
Scrieți numărul de 2 ori pentru a obține un număr de 6 cifre (ex: 347347)
Împărțiți-l la 13
Împărțiți-l la 11
Împărțiți-l la 7
Ar trebui să obțineți numărul ales inițial.
STOP

Exemplu: Pseudocod pentru coacerea unei prăjituri

Pseudocodul pentru coacerea unei prăjituri este prezentat mai jos:

Încălziți cuptorul la 180^0C
Amestecați ingredientele
Introduceți prăjitura în cuptor
Verificați dacă prăjitura este coaptă sau nu
Așteptați
    Scoateți prăjitura din cuptor
STOP

Pseudocodul este scris utilizând structura și anumite convenții din limbajele de programare. Cuvântul 'Așteptați' este **indentat**, ceea ce indică că această acțiune are loc doar dacă prăjitura nu este coaptă.

3.5 REMEDIEREA ERORILOR DIN ALGORITMI

Concepte

La scrierea algoritmilor, este ușor să faceți greșeli precum omiterea anumitor pași, așezarea pașilor într-o ordine greșită sau efectuarea de decizii incorecte. Aceste erori trebuie corectate.

Regăsiți mai jos câteva erori și modul în care le puteți remedia.

1. Secvență incorectă de instrucțiuni

Scrierea instrucțiunilor într-o ordine greșită poartă numele de secvență incorectă.

Iată mai jos un algoritm pentru perierea dinților, scris în pseudocod. Instrucțiunile sunt într-o ordine greșită. Algoritmul poate fi reparat prin mutarea acțiunii 'Pune pastă pe periuță' în capul listei.

Perie dinții din dreapta sus
Perie dinții din dreapta jos
Perie dinții din stânga sus
Perie dinții din stânga jos
Pune pastă pe periuță
STOP

2. Rezultat incorect

Iată mai jos un algoritm pentru coacerea unei prăjituri și scoaterea ei din cuptor doar atunci când este coaptă și nu înainte.

Încălziți cuptorul la 180^0C
Amestecați ingredientele
Introduceți prăjitura în cuptor
Prăjitura este coaptă?
Așteptați
Scoateți prăjitura din cuptor
STOP

Decizia de a aștepta ar putea fi greșită întrucât nu există niciun test de verificare a gradului de coacere a prăjiturii. Pentru a corecta un posibil rezultat incorect, este necesară o linie în plus în cadrul pseudocodului:

Încălziți cuptorul la 180^0C
Amestecați ingredientele
Introduceți prăjitura în cuptor
Testați cu furculița
Prăjitura este coaptă?
    Așteptați
        Scoateți prăjitura din cuptor
        STOP

3. Lipsa unui element din program

Iată mai jos un algoritm pentru coacerea unei prăjituri, din care lipsesc anumiți pași importanți.

Încălziți cuptorul la 180^0C.
Așteptați să se încingă cuptorul.
Scoateți prăjitura din cuptor.
STOP

Algoritmul poate fi remediat prin adăugarea paşilor lipsă:

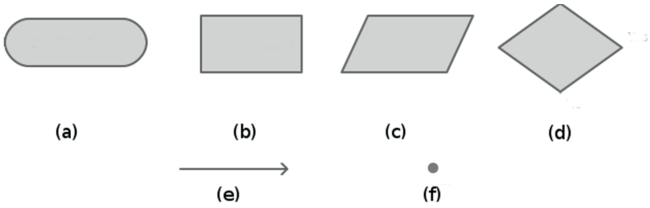
Încălziți cuptorul la 180^0C.
Amestecați ingredientele
Puneți ingredientele în tava de copt
Așteptați să se încingă cuptorul
Introduceți prăjitura în cuptor
Așteptați să se coacă prăjitura
Scoateți prăjitura din cuptor
STOP

3.6 EXERCIȚII RECAPITULATIVE

1. O secvență reprezintă:
   a. Un număr de instrucțiuni ce pot fi executate în orice ordine.
   b. Un număr de instrucțiuni ce ar trebui executate în ordine, una după cealaltă.
   c. O analiză a unei probleme ce trebuie rezolvată.
   d. O colecție de recomandări pentru îmbunătățirea unui program.

2. Care dintre următoarele nu reprezintă un algoritm?
   a. Un program.
   b. O schemă logică.
   c. Un pseudocod.
   d. Pseudoștiința.

3. Potriviți următoarele simboluri cu numele lor:



Săgeată	
Decizie	
Start sau Stop	
Proces	
Intrare sau ieșire	
Conector	

4. În pseudocodul de mai jos, ce instrucțiune este executată imediat după ‘Amestecați ingredientele’?

Încălziți cuptorul la 180^0C
Amestecați ingredientele
Introduceți prăjitura în cuptor
Verificați dacă prăjitura este coaptă și dacă nu este
    Așteptați
Scoateți prăjitura din cuptor

STOP

   a. STOP
   b. Așteptați
   c. Scoateți turta dulce din cuptor
   d. Introduceți prăjitura în cuptor

5. Următorul program trebuie să evacueze apa dintr-o mașină de spălat. Ce eroare conține?

Pornește pompa de evacuare a apei
Verifică nivelul apei
Dacă nu mai există apă
    Așteptați
Opriți pompa de evacuare
STOP

   a. Instrucțiune lipsă
   b. Secvență incorectă
   c. Rezultat incorect
   d. Indentare incorectă

CAPITOLUL 4 - PROGRAMARE

La finalul acestui capitol, veți putea să:

- Porniți și să rulați un program
- Introduceți cod într-un program
- Creați și să salvați un program
- Deschideți și să rulați un program

4.1 INTRODUCERE ÎN PYTHON

Concepte

Acest material utilizează Python, un limbaj de programare flexibil și utilizat pe scară largă, cu ajutorul căruia se pot crea programe simple sau complexe. Python rulează mai repede decât majoritatea limbajelor de programare. În plus, Python oferă un set complet de comenzi și instrucțiuni.

Acest material utilizează mediul de programare Python IDLE, care permite scrierea, editarea și rularea de cod, precum și salvarea programelor sub forma unor fișiere ce pot fi utilizate ulterior. El încorporează și un interpretor - Python Shell.

Interpretorul - Python Shell permite editarea și executarea instucțiunilor în mod interactiv, ceea ce înseamnă că în momentul în care utilizatorul introduce o instrucțiune, sistemul o evaluează, o execută și afișează rezultatul

După lansarea în execuție a Python IDLE, pe ecan apare un cursor pentru a demonstra faptul că utilizatorul poate introduce instrucțiuni. Cursorul constă în 3 caractere >>>. În momentul apariției acestora pe ecran, puteți începe tastarea codului.

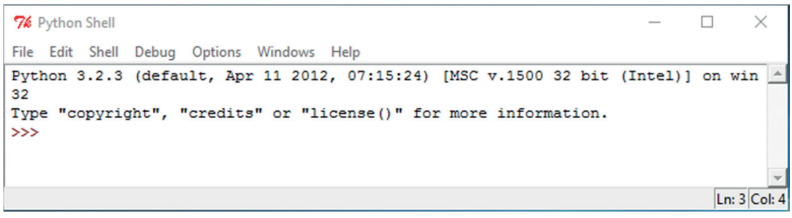
4.2 EXPLORARE PYTHON

Concepte

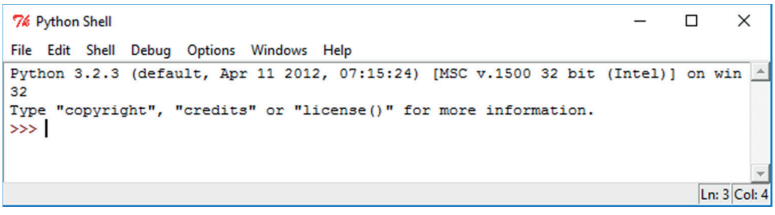
Lansare Python

1. Localizați pictograma IDLE Python aflată pe ecranul de lucru (Desktop).

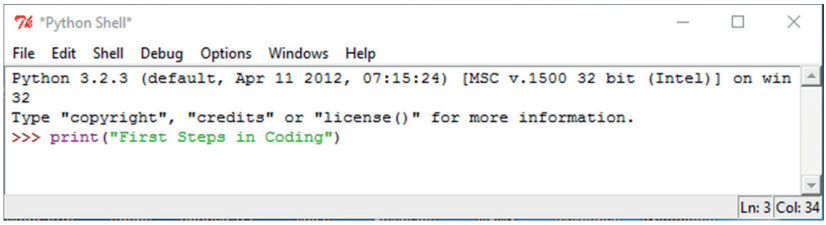
2. Executați dublu click pe pictogramă. Pe ecran va apărea o fereastră interactivă, așa cum este prezentat mai jos:



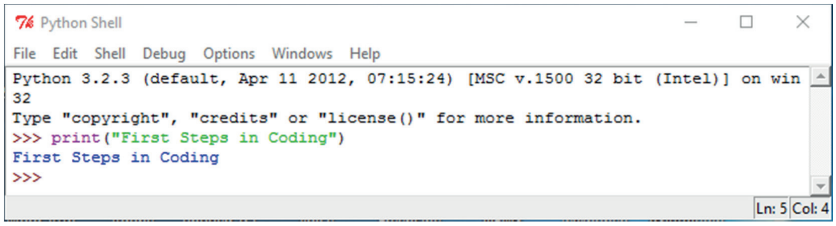
3. Executați click stânga după cursorul >>>. Acesta reprezintă semnalul că puteți introduce cod sau intrucțiuni pe care Python să le ruleze.



4. Tastați **print(“First Steps in Coding”)**



5. Apăsați tasta **Enter**.



6. Vizualizați rezultatul.
7. Apăsați butonul **X** pentru a închide fereastra.

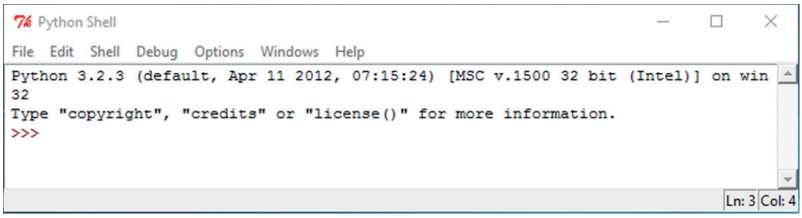
Cum funcționează programul

- În exemplul anterior, Python a interpretat și executat codul care îi indica să afișeze pe ecran textul aflat între ghilimele.
- Rezultatul este afișarea pe ecran a textului **First Steps in Coding**.
- În Python, comanda print() este utilizată pentru a afișa informații pe ecran.

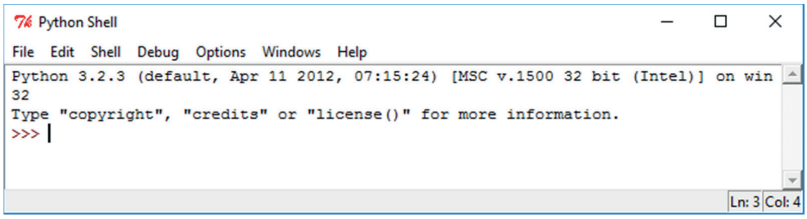
Să încercăm altă instrucțiune:

1. Localizați pictograma IDLE Python aflată pe ecranul de lucru (Desktop).

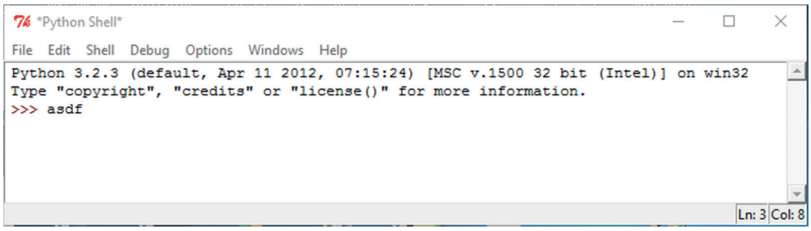
2. Executați dublu click pe pictogramă. Pe ecran va apărea o fereastră interactivă, așa cum este prezentat mai jos:



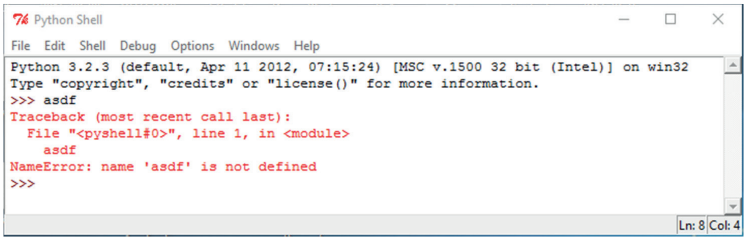
3. Executați click stânga după cursorul >>>. Acesta reprezintă semnalul că puteți introduce cod sau intrucțiuni pe care Python să le ruleze.



4. Tastați **asdf**



5. Apăsați tasta **Enter**
6. Vizualizați rezultatul. În acest exemplu, Python nu înțelege literele **asdf**. Observați cele 4 rânduri scrise cu culoarea roșie. Python afișează **mesaje de eroare** atunci când nu înțelege textul introdus de utilizator. În mod normal, ultima linie a mesajului de eroare conține cele mai utile informații despre eroarea respectivă, în acest caz " 'asdf' is not defined " (" 'asdf' nu este definit ").



7. Apăsați butonul **X** pentru a închide fereastra.

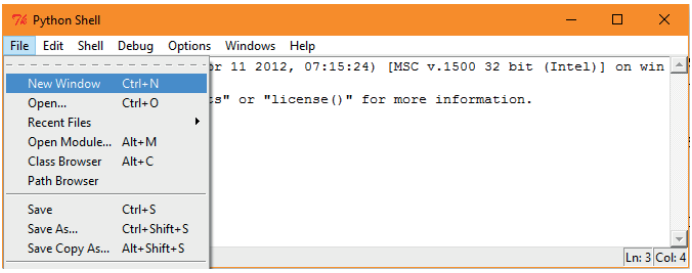
4.3 SALVAREA UNUI PROGRAM

Concepte

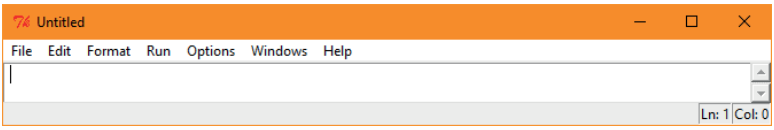
Atunci când scrieți un cod, este recomandabil să îl salvați astfel încât să îl puteți reutiliza ulterior. Extensia fișierelor Python este **.py**. Această extensie anunță calculatorul că acest fișier reprezintă un program Python, ca de exemplu **MagicTrick.py**.

Crearea și salvarea unui program

1. Deschideți aplicația Python. Executați click pe meniul **File, New Window**.

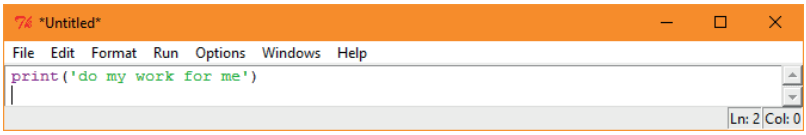


2. Este creată o nouă fereastră, fără denumire. Executați click în zona mare albă a ferestrei.



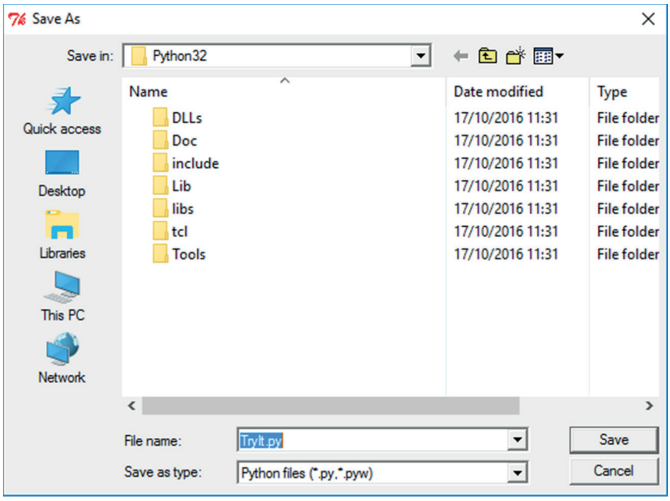
3. Introduceți următorul text și apoi apăsați tasta Enter.

print('do my work for me')

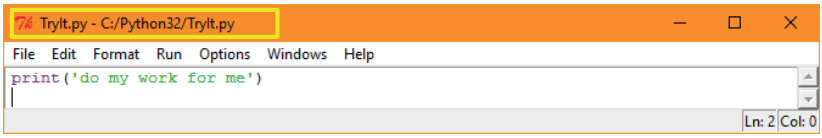


4. Executați click pe meniul **File, Save As** și observați apariția unei ferestre de dialog. Remarcați următoarele:

   - Locația implicită de salvare a fișierelor Python este Python 32. Pentru moment poate fi folosită aceasta, însă o puteți modifica, în funcție de dorințele dvs.
   - Tipul de fișier implicit este **Python files (*py,*pyw)**



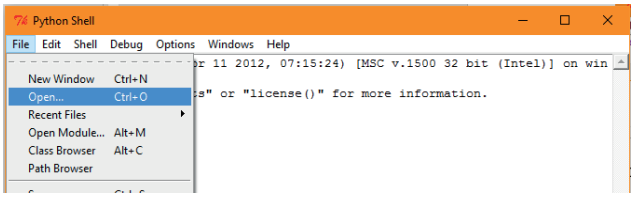
5. În caseta **File name**, tastați numele fișierului **TryIt.py**.
6. Executați click pe butonul **Save**.



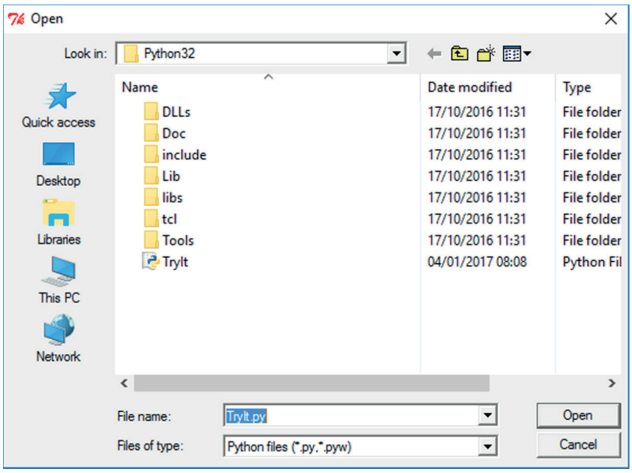
7. Remarcați că numele ferestrei va fi modificat în '**TryIt.py'**. Acesta indică numele programului sau fișierului.
8. Apăsați butonul **X** pentru a închide fereastra și programul.

Deschiderea și rularea unui program existent

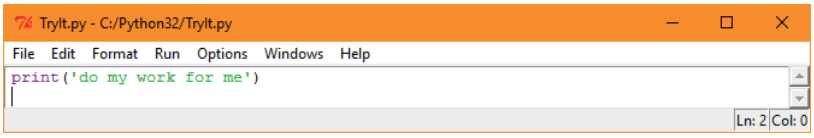
1. Deschideți aplicația Python
2. Executați click pe meniul **File, Open.**



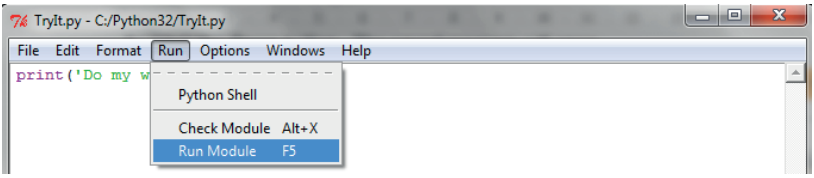
3. În fereastra apărută pe ecran, selectați numele fișierului "**TryIt.py**" sau tastați-l în caseta **File name**.



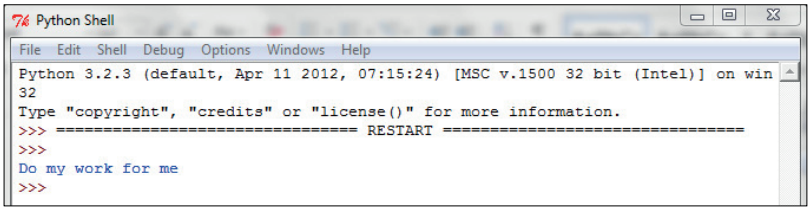
4. Apăsați butonul **Open**. Programul salvat se va deschide.



5. Pentru rularea programului executați click pe meniul **Run** existent în bara de meniu și apoi selectați opțiunea **Run Module**:



6. La rularea programului, iată ce ar trebui să se afișeze pe ecran.



7. O altă modalitate de a rula un program este prin apăsarea tastei **F5** . Închideți fereastra de rezultat prin apăsarea butonul **X** aflat în colțul din dreapta sus. Reveniți în program și apăsați tasta **F5** pentru rularea programului. Vizualizați rezultatele.

4.4 EXERCIȚII RECAPITULATIVE

1. Cum rulați un program în Python?

   a. Tastați textul **run**.
   b. Apăsați tasta **F5**.
   c. Tastați textul **go**.
   d. Tastați textul **start**.

CAPITOLUL 5 -
EFECTUAREA CALCULELOR

La finalul acestui capitol, veți putea să:

- Recunoașteți și să utilizați operatorii aritmetici +,-,* și /
- Cunoașteți modul în care parantezele afectează evaluarea expresiilor matematice
- Înțelegeți și aplicați ordinea operatorilor în cadrul expresiilor complexe
- Înțelegeți cum să utilizați parantezele pentru a structura expresiile complexe

5.1 EFECTUAREA CALCULELOR ÎN PYTHON

Concepte

Operatori

Ca și majoritatea limbajelor de programare, Python poate efectua calcule matematice sau poate evalua expresii matematice, cum ar fi:

10+12+15.

El utlizează simbolul ***** pentru înmulțire și simbolul / pentru împărțire în locul simbolurilor **x** și ÷.

NOTAȚIE	SEMNIFICAȚIE
3 * 4	3 înmulțit cu 4
3 / 4	3 împărțit la 4
3 + 4	3 plus 4
3 - 4	3 minus 4

În Python:

7 înmulțit cu 9 este scris 7 * 9
63 împărțit la 3 este scris 63 / 3

Spațiile dintre numere sunt opționale și 7*9 va funcționa la fel de bine ca și 7 * 9.

Simbolurile matematice *, /, + și -, utilizate în programare pentru efectuarea calculelor poartă numele de **operatori**.

Paranteze

Expresiile matematice în Python pot conține de asemenea paranteze, ca de exemplu:

10-(6-4)

Dacă evaluăm expresia de la stânga la dreapta, ignorând parantezele, soluția ar fi 0.

10-(6-4) = 0

Totuși, acest răspuns este incorect.

Parantezele indică ce anume trebuie calculat întâi. În Python, ca și în matematică, ceea ce este trecut între paranteze este calculat mai întâi. Pentru evaluarea unei expresii de felul 10-(6-4), Python o divizează astfel:

Întâi: 6-4=2

Apoi: 10-2=8

Deci, 10-(6-4)=8

5.2 ORDINEA OPERATORILOR

Concepte

Atunci când există paranteze în cadrul unei expresii, este clar pentru Python în ce ordine trebuie să calculeze expresia respectivă. Atunci când nu există paranteze, există o ordine acceptată a operațiilor. O regulă a limbajului formal determină ordinea de aplicare a operatorilor. Această regulă poartă denumirea de **Prioritatea Operatorilor**. În majoritatea limbajelor de programare, secvența sau ordinea în care operatorii sunt aplicați este înmulțire, împărțire, adunare, scădere.

Operatorii * și / sunt aplicați înaintea operatorilor + și -.

Iată mai jos câteva exemple:

1+7+2+3

1+7 = 8, 8+2 = 10, 10+3 = 13 răspuns corect: 13

2*2*3+4

2*2 = 4, 4*3 = 12, 12+4 = 16 răspuns corect: 16

4+3*2=?

Ați putea crede că răspunsul corect este 14, pentru că:

4+3 = 7, 7*2 = 14

Totuși, având în vedere ordinea operatorilor, înmulțirea este efectuată prima. 4+3*2 este calculată exact ca și cum ar fi fost scrisă sub forma 4+(3*2). Ca urmare:

4+(3*2) = 4+6, și 4+6 = 10 răspuns corect: 10

31*7+112*2

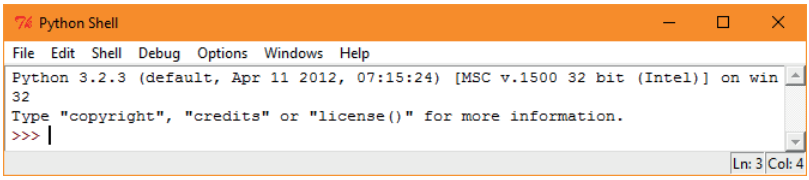
Cele 2 înmulțiri sunt efectuate primele și apoi operația de adunare.

31*7+112*2 = 217+224 și 217+224 = 441

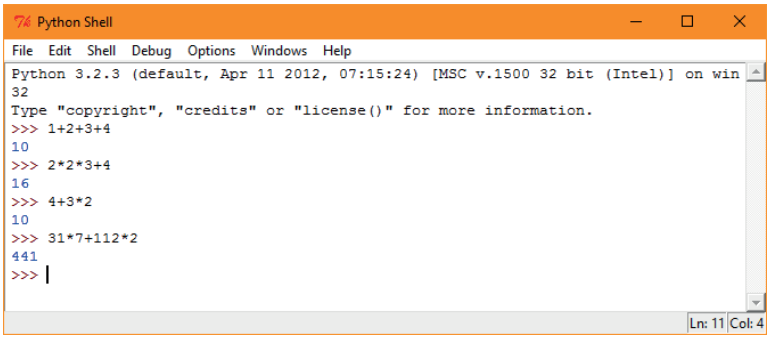
Utilizarea Python ca și Calculator

Efectuați aceleași calcule ca în secțiunea precedentă, dar de această dată utilizând Python Shell:

1. Deschideți aplicația Python. Executați click după cursorul >>>.



2. Tastați expresiile (calculele). Apăsați tasta **Ente**r după ce ați terminat de introdus fiecare expresie.



3. Verificați rezultatele.

Exemplu: Calcularea costurilor pentru găleți cu vopsea

Dacă 1 litru de vopsea roșie costă 31€ și 1 litru de vopsea aurie costă 112€, atunci costul a 7 litri de vopsea roșie și 2 litri de vopsea aurie poate fi calculat astfel:

31*7+112*2

În acest caz, se poate observa că este important să se calculeze întâi costul pe fiecare culoare și apoi să se obțină costul total.

Așadar, (31*7)+(112*2) oferă prețul total corect.

Parantezele nu sunt necesare în acest caz datorită ordinii operațiilor.

Totuși, puteți adăuga oricând paranteze suplimentare în cadrul unei expresii pentru a modifica ordinea operațiilor. Parantezele au prioritate asupra oricărei reguli legate de ordinea operațiilor.

5.3 EXERCIȚII RECAPITULATIVE

1. Cum convingeți Python să evalueze o expresie?
   a. Tastați 'evaluate', și apoi scrieți expresia.
   b. Divizați întâi expresia în părți mai mici.
   c. Tastați expresia după cursorul >>>.
   d. Folosiți mai curând un calculator de birou.

2. Care dintre următoarele expresii este utilizată pentru a înmulți numerele 4 și 7 în Python?
   a. 4 + 7
   b. 4 x 7
   c. 4 # 7
   d. 4 * 7

3. Evaluați expresia 3*4*2+8 utilizând regula de Ordine (Prioritate) a Operatorilor și selectați de mai jos răspunsul corect:
   a. 24
   b. 32
   c. 104
   d. 120

CAPITOLUL 6 –
VARIABILE ȘI TIPURI DE DATE

La finalul acestui capitol, veți putea să:

- Utilizați diferite tipuri de date: șir de caractere (string), caracter (character), număr întreg (integer), număr real (float), boolean
- Definiți termenul de variabilă și să evidențiați scopul unei variabile în cadrul unui program.
- Definiți și inițializați o variabilă
- Atribuiți o valoare unei variabile
- Utilizați date introduse de utilizator într-un program
- Utilizați datele de ieșire dintr-un program, afișate pe monitor

6.1 TIPURI DE DATE

Concepte

În programare sunt utilizate toate tipurile de date pentru rezolvarea problemelor și generarea informațiilor. Tipul de date determină modul de stocare a acestora în memoria calculatorului, precum și operațiile care pot fi efectuate cu datele respective. Spre exemplu, vârsta unei persoane este stocată sub forma unui număr și se pot efectua diverse calcule matematice pe baza ei, ca de exemplu compararea ei cu vârsta altor persoane. Iar numele unei persoanei este stocat ca și text.

În Pyhon **tipurile de date** utilizate cel mai frecvent sunt: **număr întreg (integer)**, **număr real (float)**, **șir de caractere (string)** și **Boolean**.

Majoritatea limbajelor de programare au tipuri de date standard, care diferă doar ca denumire. De exemplu, tipul de date 'string' (șir de caractere) utilizat în Python este folosit în alte limbaje de programare ca și tip de date 'character' (caracter). Tipul de date character poate fi utilizat de asemenea pentru a defini litere sau caractere precum 'a' 'b' 'c' '@' etc.

Tipuri de date în Python

Număr întreg (integer)

Descrie sau definește un număr întreg de orice dimensiune, pozitiv sau negativ, precum 1, 3, 9,999,999, -712 sau 0.

Număr real (float)

Descrie sau definește un număr zecimal precum 11.456, -71.3, 29.0.

Numerele reale sunt utilizate de asemenea pentru a reprezenta numere foarte mari sau foarte mici în notații științifice. De exemplu, 5.0e30, ceea ce înseamnă 5 urmat de 30 de zerouri, sau 1.7e-6 ceea ce înseamnă 0.000017, adică 1.7 împărțit la 1 urmat de 6 zerouri.

Șir de caractere (string)

Descrie sau definește o porțiune de text sau un 'șir' de caractere. De exemplu, numele unei persoane, precum 'John Smith'. Valorile de tip String apar de obicei ca urmare a utilizării comenzii input().

Boolean

Descrie sau definește o valoare ce poate fi fie Adevărată, fie Falsă. Tipurile de date Booleene pot avea întotdeauna doar aceste 2 valori. Valorile booleene sunt generate de obicei ca rezultat a comparației numerelor între ele.

De exemplu expresia:
3 < 4 returnează valoarea booleană True.

6.2 VARIABILE

Concepte

O **variabilă** este utilizată în cadrul unui cod pentru a reprezenta anumite date astfel încât acestea să poată fi ulterior folosite de mai multe ori în cadrul unui program. O variabilă reprezintă un înlocuitor al valorilor actuale. Ea poate stoca valoarea și o poate păstra pentru o utilizare ulterioară.

Atribuirea unei valori unei variabile

O variabilă poate reprezenta diferite tipuri de date sau valori, cum ar fi un număr sau un șir de caractere. În cadrul codului trebuie să specificați ce valoare va reprezenta variabila – acest lucru este cunoscut sub denumirea de atribuirea unei valori unei variabile.

În exemplul următor, valoarea 3 este atribuită variabilei numite x.

x=3

În acest caz, de fiecare dată când variabila **x** va fi utilizată în cadrul programului, ea va reprezenta valoarea **3**.

O variabilă poate stoca de asemenea și un șir de caractere, adică o colecție de caractere. În exemplul următor, valoarea **Good Morning**, ceea ce reprezintă un șir de caractere, este atribuită variabilei numite **Greeting**.

Greeting = 'Good Morning'

În acest caz, de fiecare dată când variabila **Greeting** va fi utilizată în cadrul programului, ea va reprezenta **Good Morning**. După atribuire, variabila numită Greeting poate fi utilizată în locul șirului de caractere 'Good Morning'.

Utilizarea comenzii print() pentru afișarea unui rezultat

Python afișează pe ecran conținutul unei variabile, utilizând comanda **print()**.
Așadar, în exemplul x=3
Comanda **print(x)** va afișa pe ecran valoarea **3**.
Comanda **print(x+2)** va afișa pe ecran valoarea **5**.
Și în exemplul Greeting = 'Good Morning'
Comanda **print(Greeting)** va afișa pe ecran valoarea **'Good Morning'**.

Definirea, Inițializarea și Actualizarea Variabilelor

Definirea Variabilelor

Definirea unei variabile înseamnă definirea în cadrul codului a tipului de date pe care respectiva variabilă îl va stoca, cum ar fi un număr (e.g. întreg, real) sau un șir de caractere. În Python, acest lucru se realizează prima dată când se atribuie o valoare unei variabile. Python determină faptul că variabila va stoca tipul de date al valorii atribuite.

Dacă se asociază un număr (i.e. valoare) variabilei x, Python va presupune că x va reprezenta întotdeauna o valoare numerică.

x=3

Dacă atribuiți șirul de caracatere 'country' variabilei q, Python va ști că q va reprezenta întotdeauna un șir de caractere. Textul se încadrează între ghilimelele simple pentru a indica că este vorba de un șir de caractere.

q='country'

Orice caractere incluse între ghilimele simple sunt considerate un șir de caractere, chiar și un număr. În exemplul de mai jos, 12 este un șir de caractere și nu un număr:

Age = '12'

Inițializarea Variabilelor

Înainte de utilizarea unei variabile, trebuie să îi atribuiți o valoare inițială. Prima dată când se atribuie o valoare unei variabile indică **inițializarea variabilei** respective.

Dacă încercați să aplicați o comandă unei variabile înaintea ca aceasta să fi fost inițializată, veți primi un mesaj de eroare. Înainte ca o valoare să fie atribuită unei variabile, variabila este **neinițializată**.

Actualizarea Variabilelor

Valoarea unei variabile poate fi actualizată prin atribuirea unei noi valori acesteia. Veți actualiza variabila și automat ea va fi actualizată peste tot în cadrul programului, unde a fost utilizată. Acesta este un motiv principal pentru care se utilizează variabilele întrucât este suficientă o singură actualizare care să modifice mai multe apariții ale variabilei respective. O variabilă stochează întotdeauna cea mai recentă valoare atribuită acesteia.

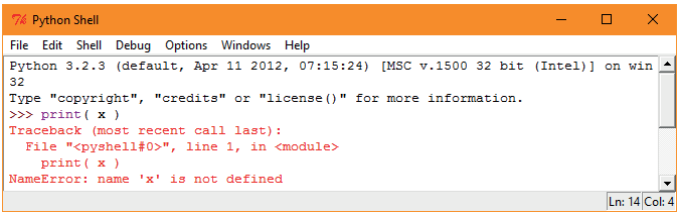
Se consideră variabila x cu valoarea curentă 3. Valoarea atribuită poate fi modificată în 7 prin introducerea codului:

x=7

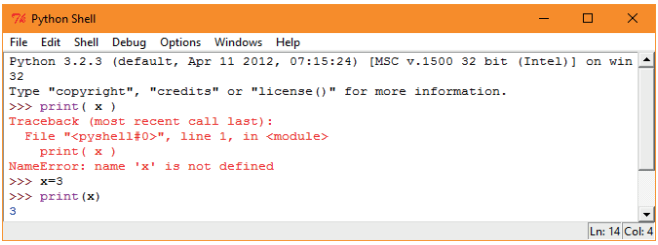
Comanda **print(x)** va afișa acum valoarea **7**.
Comanda **print(x+2)** va afișa acum valoarea **9**.

Exemplu: Lucrul cu variabile

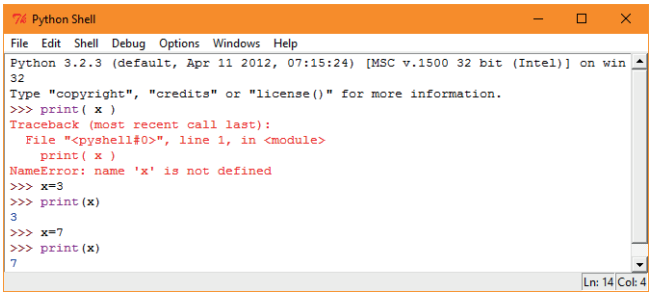
1. Deschideți aplicația Python.
2. Înaintea inițializării variabilei x, scrieți comanda print(x).
3. Observați mesajul de eroare.



4. Inițializați variabila x cu valoarea 3.
5. Introduceți din nou comanda print(x).



6. Actualizați variabila x la valoarea 7.
7. Afișați noua valoare.



6.3 DINCOLO DE NUMERE

Concepte

Lucrul cu șiruri de caractere

Exact la fel cum adunăm numere putem alătura caractere sau șiruri de caractere. Pentru a face acest lucru în Python, puteți folosi simbolul +.

Spre exemplu, inițializați variabila **Name** cu valoarea **"David"** și variabila **Question** cu valoarea **"What is your age?"**

Name= "David"

Question = "What is your age "+Name+"?"

Acest lucru va returna următorul rezultat:

'What is your age David?'

Remarcați spațiile libere în cadrul șirului de caractere. Dacă nu se adăuga spațiu la finalul șirului de caractere, atunci nu ar fi existat niciun spațiu între acesta și numele persoanei, iar întrebarea ar fi arătat astfel:

'What is your ageDavid?'

Modificarea șirurilor de caractere în numere

Uneori, un șir de caractere trebuie convertit într-un număr (e.g. întreg sau real). De exemplu, dacă variabila Age este șirul de caractere '12',

Age = '12'

atunci Age +1 va retuna o eroare, întrucât nu se poate adăuga un număr la un șir de caractere.

Pentru remedierea acestei erori, puteți converti variabila Age într-un număr întreg astfel:

Age = int(Age)

Utilizarea comenzii input() pentru introducerea datelor de către utilizator

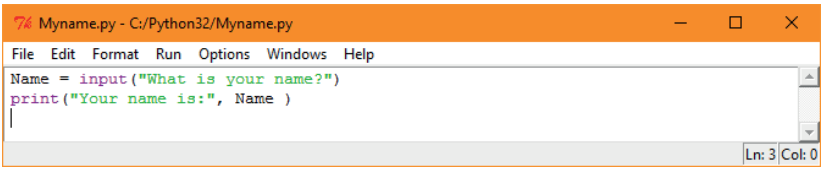
Puteți scrie un cod care să solicite utilizatorului să introducă anumite informații sub forma unui șir de caractere, ca de exemplu numele lor.

În Python puteți utiliza comanda **input()** pentru a ruga utilizatorul să introducă un șir de caractere.

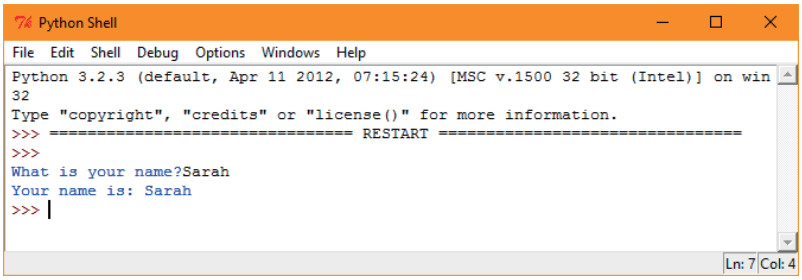
Comanda **input()** este o instrucțiune în Python care solicită utilizatorului să introducă o anumită informație, iar programul așteaptă până când informația este introdusă.

Exemplu: Introducerea unui nume și afișarea lui pe ecran.

1. Deschideți aplicația Python și creați un nou fișier.
2. Introduceți următorul cod.



3. Rulați programul.
4. La întrebarea "What is your name?" răspundeți tastând numele dvs.
5. Apăsați tasta Enter.
6. Salvați programul cu numele **Myname.py**.



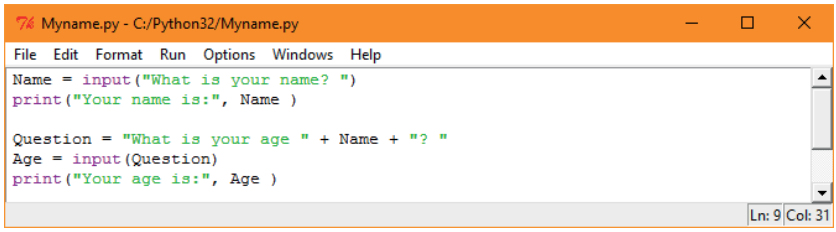
7. Executați click pe butonul X pentru a închide fereastra.

Cum funcționează programul

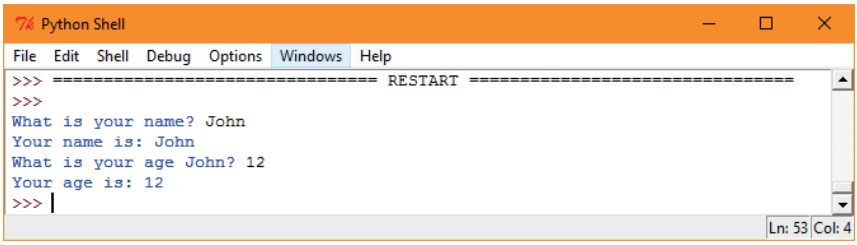
- Utilizatorul este rugat să introducă o valoare, în acest caz numele lui, prin utilizarea comenzii **input()**.
- Programul așteaptă introducerea informațiilor de către utilizator.
- Variabila 'Name' este inițializată pentru a stoca numele introdus de utilizator.
- Comanda **print()** este utilizată pentru a afișa pe ecran textul "Your name is:" urmat de conținutul variabilei 'Name'.

Exemplu: Lucrul cu șiruri de caractere

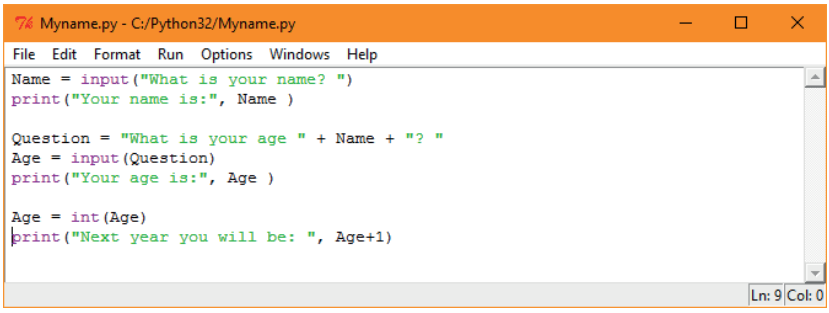
1. Deschideți fișierul salvat anterior, MyName.py, utilizând pașii învățați anterior.
2. Modificați exemplul pentru a arăta ca mai jos:



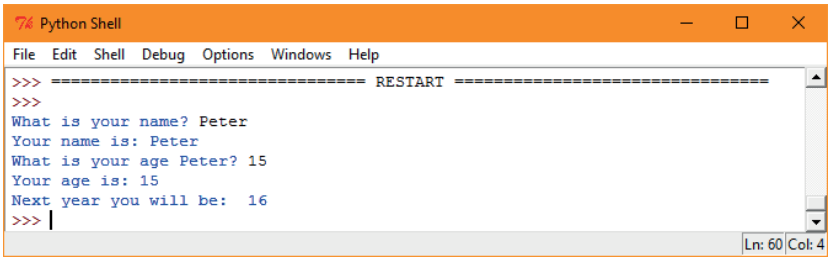
3. Rulați programul.
4. Răspundeți la prima întrebare și apăsați tasta Enter.
5. Răspundeți la a doua întrebare și apăsați tasta Enter.



6. Modificați programul astfel



7. Rulați programul din nou.
8. Răspundeți la cele 2 întrebări.



9. Salvați și închideți fișierul.

6.4 EXERCIȚII RECAPITULATIVE

1. Care dintre următoarele afirmații despre variabile este corectă?
    a. Variabilele nu își modifică niciodată valoarea.
    b. Numele variabilei trebuie să fie scris doar cu litere mici.
    c. Variabilele trebuie inițializate cu valoare 0.
    d. Uneori o variabilă poate stoca o valoare booleană.

2. Care este scopul unei variabile?
    a. Pentru a modifica mai ușor programul pe viitor.
    b. Pentru a-i arăta aplicației Python cum să combine 2 numere.
    c. Pentru stocarea anumitor valori astfel încât acestea să poată fi reutilizate ulterior în cadrul programului.
    d. Pentru asigurarea unei indentări corecte a codului sursă.

3. Care dintre următoarele instrucțiuni Python inițializează o variabilă?
    a. Initialise().
    b. x=3.
    c. start().
    d. finish().

4. În Python, dacă o variabilă numită 'length' are valoarea 200 și instrucțiunea 'length=400' este executată, ce se va întâmpla?
    a. Python va returna un mesaj de eroare, întrucât nu se poate schimba valoarea unei variabile.
    b. Python va returna un mesaj de eroare, întrucât nu puteți schimba valoarea 200 în 400.
    c. Python va atribui valoarea 400 variabilei length.
    d. Python va adăuga 400 la valoarea variabilei length, aceasta având acum valoarea 600.

5. În Python, funcția input():
    a. Este utilizată pentru a conecta la computer un stick de memorie
    b. Poate fi utilizată cu un cursor pentru a introduce datele mai ușor sau fără cursor pentru a introduce datele mai dificil.
    c. Returnează un șir de caractere.
    d. Trebuie să fie prima funcție apelată în cadrul unui program.

6. În Python, funcția print()
    a. Este utilizată dacă se dorește schimbarea tipului de date
    b. Este utilizată pentru afișarea unui rezultat
    c. Este ultima comandă din cadrul unui program.
    d. Trebuie să fie prima funcție apelată în cadrul unui program.

7. Un tip de date care conține doar 2 valori poartă numele de
    a. Întreg (Integer)
    b. Real (Float)
    c. Șir de caractere (String)
    d. Boolean

CAPITOLUL 7 –
ADEVĂRAT SAU FALS

La finalul acestui capitol, veți putea să:

- Utilizați expresii Booleene în cadrul unui program
- Recunoașteți tipurilor de expresii logice booleene pentru a genera o valoare adevărată sau falsă: =, >, <, >=, <=, <>, !=, ==, AND, OR, NOT
- Înțelegeți ordinea operatorilor și ordinea de evaluare a expresiilor complexe

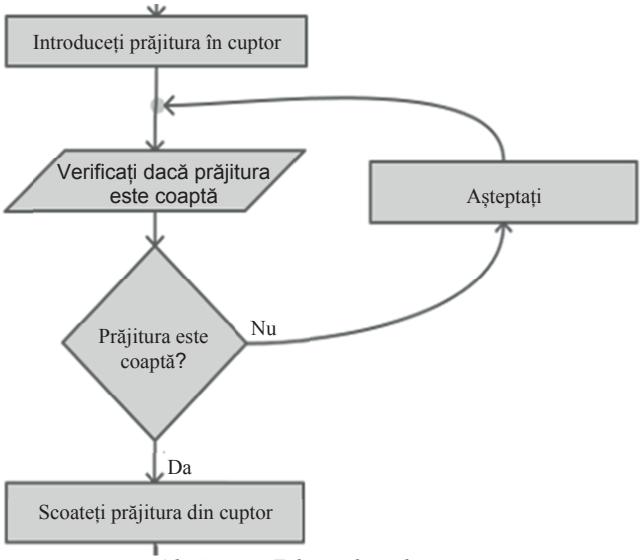
7.1 EXPRESII BOOLEENE

Concepte

În viața de zi cu zi, verificați adesea dacă o informație este **Adevărată** sau **Falsă** înainte de luarea unei decizii. Acest concept este folosit de asemenea și în cadrul programelor de calculator, unde se verifică dacă anumite condiții sunt **Adevărate** sau **False** înaintea stabilirii pașilor următori**.** De exemplu:

Este frig afară?
Adevărat (True) – ia-ți o haină
Fals (False) – lasă haina acasă

Este coaptă prăjitura?
Adevărat (True) – scoate-o din cuptor
Fals (False) – mai coace-o încă 5 minute și apoi verifică iar



Adevărat sau Fals în scheme logice

În cadrul unei scheme logice, verificările sau testările sunt reprezentate prin blocuri de decizie cu rezultat **Da (Yes)** sau **Nu (No)**.

Programele lucrează cu rezultate **Adevărat (True)** și **Fals (False)** decât cu **Da (Yes)** și **Nu (No)**.True și False reprezintă concepte importante în programare întrucât este necesară o modalitate de a testa dacă anumite condiții sunt îndeplinite pentru a putea controla pașii următori în cadrul programului.

În programare, acest test logic poartă denumirea de **expresie Booleană**. O **expresie booleană** determină o **valoare booleană** ce poate fi fie adevărată, fie falsă.

O modalitate prin care Python utilizează expresiile booleene este să compare dimensiunea unei valori numerice cu altă valoare și să returneze un rezultat de tip 'True' sau 'False'. El execută acest lucru utilizând **un operator de comparație**. Acesta compară cele 2 valori între ele și decide relația lor. Uneori, aceștia mai poartă numele de operatori relaționali.

$$99 > 7$$

Aici, operatorul '>' compară valorile 99 și 7 pentru a vedea dacă valoarea din stânga este mai mare decât valoarea din dreapta. Dacă acest lucru este adevărat, el va returna rezultatul True, în caz contrar va returna rezultatul False.

Așadar: 99 > 7 este Adevărat (True) iar 6 > 7 este Fals (False).

7.2 OPERATORI DE COMPARAȚIE

Concepte

Operatorii de comparație reprezintă un tip de expresii logice Booleene utilizate pentru a compara valori și a decide dacă rezultatul este adevărat sau fals.

Tabel al operatorilor de comparație

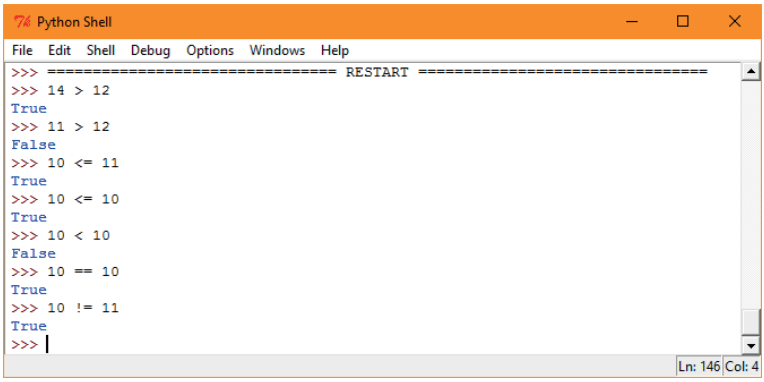
Există 6 operatori de comparație. În tabelul de mai jos X și Y sunt variabile ce stochează valori numerice:

NOTAȚIE	SEMNIFICAȚIE
X > Y	X este mai mare decât Y
X < Y	X este mai mic decât Y
X >= Y	X este mai mare sau egal cu Y
X <= Y	X este mai mic sau egal cu Y
X == Y	X este egal cu Y
X != Y	X nu este egal cu Y

Notă: Unele limbaje de programare pot folosi operatorul <> pentru a desemna inegalitatea dintre 2 valori și = pentru a desemna egalitatea. Python 3.x nu folosește <> și folosește == în loc de =.

Exemplu: Operatori de comparație

1. Deschideți aplicația Python.
2. Introduceți exemplele de mai jos pentru a verifica că ați înțeles cum se folosesc operatorii de comparație.



3. Construiți-vă propriile exemple.

7.3 OPERATORI BOOLEENI

Concepe

Operatorii booleeni sunt cuvintele **și (and)**, **sau (or)**, **negație (not)**, care sunt utilizate pentru a combina expresii booleene simple în scopul obținerii unui rezultat final de tip boolean.

Există 3 tipuri de operatori Booleeni de bază:

and

Combină 2 valori booleene și returnează un rezultat – adevărat dacă ambele valori sunt adevărate și fals în caz contrar

Expresie	*Rezultat*
True and True	True
True and False	False
False and True	False
False and False	False

or

Combină 2 valori booleene și returnează un rezultat - Adevărat dacă una dintre valori este adevărată sau dacă ambele valori sunt adevărate, Fals în caz contrar

Expresie	*Rezultat*
True or True	True
True or False	True
False or True	True
False or False	False

not

Convertește o singură valoare Booleană din Adevărat în Fals și invers.

Expresie	*Rezultat*
not True	False
not False	True

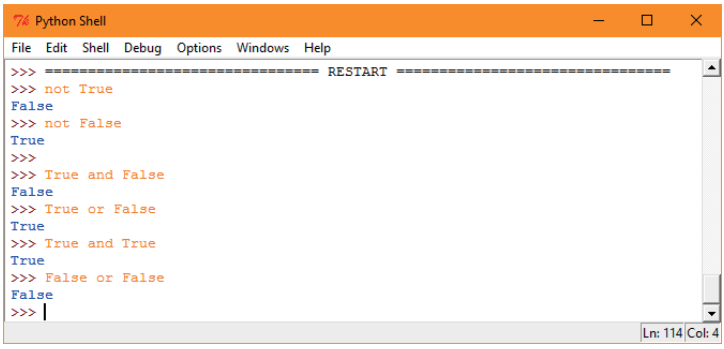
Expresiile Booleene pot utiliza atât operatorii de comparație, cât și operatorii Booleeni, ca de exemplu:

$$x < 3 \text{ and } y < 12$$

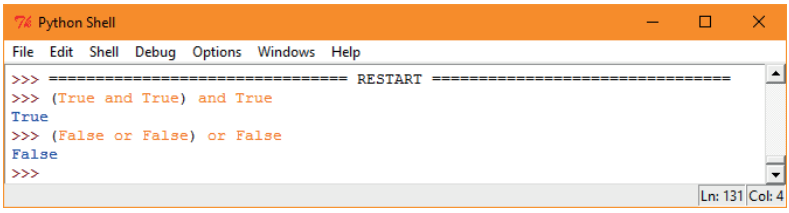
Într-o astfel de expresie mixtă, operatorii de comparație sunt evaluați înaintea operatorilor Booleeni. Acest lucru se datorează ordinii operatorilor despre care vom vorbi mai încolo.

Exemplu: Expresii Booleene

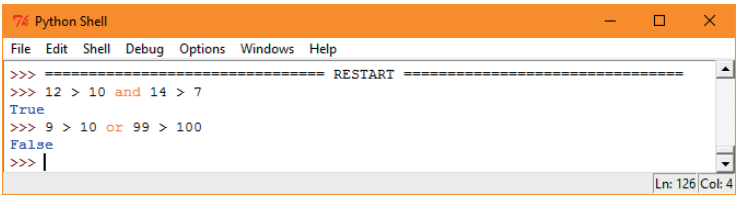
1. Deschideți aplicația Python.
2. Încercați să evaluați expresii Booleene în Python.
3. Gândiți-vă la anumite combinații pentru a vă asigura că ați înțeles scopul operatorilor **and**, **or** și **not**.



4. Construiți câteva expresii Booleene complexe folosind paranteze, ținând cont că expresiile din interiorul parantezelor sunt evaluate primele.



5. Utilizați Python pentru a evalua câteva expresii Booleene rezultate din compararea numerelor.



7.4 VARIABILE DE TIP BOOLEAN

Concepte

Inițializarea și atribuirea valorilor unor variabile se aplică pentru nume, șiruri de caractere și valori Booleene.

Instrucțiunea **Y = 200** va inițializa variabila numită Y cu valoarea numerică 200.

Instrucțiunea **A = False** va inițializa variabila numită A cu valoarea Booleană False.

Instrucțiunea **name=input("What is your name?")** va inițializa variabila numită name cu un șir de caractere obținut ca răspuns la întrebarea "What is your name?"

Iată ce se întâmplă:

i. Utilizatorul este întrebat "What is your name?"
ii. Utilizatorul își introduce numele sub forma unui șir de caractere.
iii. Valoarea acestui șir de caractere este atribuită varibilei denumite name.

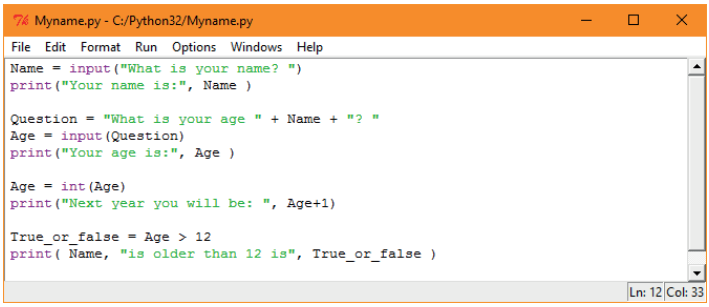
Următorul exemplu utilizează o variabilă de tip Boolean denumită True_or_false.

Remarcați folosirea simbolului _ (underscore) între cuvintele din denumirea variabilei. Python tratează acest simbol din cadrul numelui unei variabile ca pe orice altă literă. Ca urmare, True_or_false desemnează o singură variabilă. Dacă utilizați spațiu în loc de _, Python va trata denumirea variabilei ca 3 cuvinte separate și va returna o eroare.

Instrucțiunea **True_or_false = Age > 12 c**ompară variabila Age cu valoarea 12 și stochează rezultatul Boolean în cadrul variabilei True_or_false.

Exemplu: Lucrul cu valori de tip Boolean

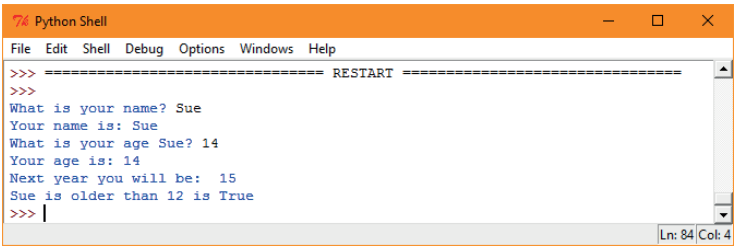
1. Deschideți programul Myname.py și modificați-l așa cum este prezentat mai jos.



2. Rulați programul.

Introduceți numele dvs.

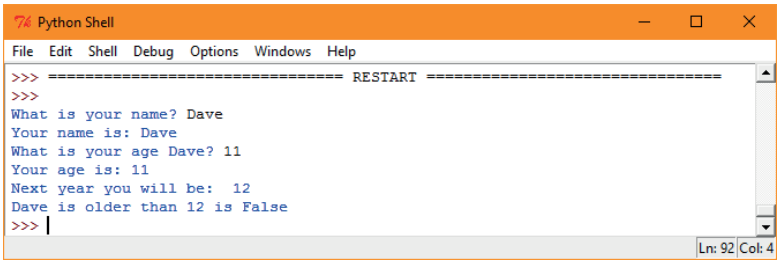
Introduceți o vârstă mai mare ca 12.



3. Rulați programul din nou.

Introduceți numele dvs.

Introduceți o vârstă mai mică ca 12 pentru a vedea rezultatul.



7.5 COMBINAREA EXPRESIILOR BOOLEENE

Concepte

Utilizarea parantezelor în expresiile Booleene

Așa cum a fost explicat și anterior, parantezele sunt importante în cadrul expresiilor numerice. Ele sunt de asemenea importante și în cadrul expresiilor Booleene.

(A and B) or C nu este același lucru ca și **A and (B or C)**

Să presupunem de exemplu că A este False, B este True și C este True. Introducerea acestor valori în 2 expresii returnează rezultate diferite:

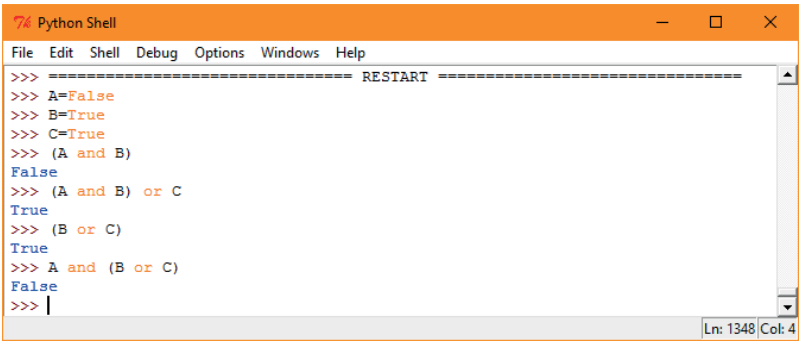
(A and B) or C este evaluată ca fiind True

A and (B or C) este evaluată ca fiind False

Pentru a verifica acest lucru, tastați expresiile Booleene în Python, așa cum este prezentat mai jos.

Exemplu: Utilizarea parantezelor în expresii Booleene

1. Deschideți aplicația Python. Tastați următoarele expresii:



Remarcați cum utilizarea parantezelor a schimbat rezultatul expresiei A and B or C.

Ordinea operatorilor, inclusiv Booleeni

Tabelul de mai jos reprezintă o listă parțială a ordinii în care sunt majoritatea operatorilor importanți sunt aplicați. În lista de mai jos, operatorii din partea de sus a listei sunt aplicați înaintea celor aflați la finalul listei.

Simbol	*Operație*
* /	Înmulțire și împărțire
+ -	Adunare și scădere
<, >, <=, >=, ==, !=,	Operatori de comparare
not	Operator Boolean not
and	Operator Boolean and
or	Operator Boolean or

Spre exemplu, în absența parantezelor, * este evaluat înainte de + (și în cadrul listei de mai sus * este înainte de +). Similar, '**and**' este evaluat înainte de '**or**' (la fel cum este evidențiat și în lista de mai sus).

O expresie precum:

A or B and C

va fi evaluată astfel:

A or (B and C)

O expresie complexă precum:

x < 2 or 1+3*x > 2+4/2 and not x < y

va fi calculată astfel:

(x < 2) or (1+(3*x) > 2+(4/2) and (not (x < y)))

Uneori chiar și programatorii experimentați uită ordinea exactă de evaluare a operatorilor. În cadrul unei expresii complexe precum cea de mai sus, este uzual să se includă anumite paranteze pentru a fi mai clară expresia de evaluat:

x < 2 or ((1+3*x > 2+4/2) and not x < y)

Ordinea operatorilor pentru operațiile *, /, + și – este utilizată atât de frecvent încât rar se utilizează parantezele pentru clarificarea ordinii de evaluare a expresiei.

7.6 EXERCIȚII RECAPITULATIVE

1. Care dintre următoarele reprezintă o expresie în Python pentru a aduna 3 numere și a împărți rezultatul la 3?
   a. 4+5+6/3
   b. 4+5+6÷3
   c. (4+5+6)/3
   d. (4+5+6)÷3

2. Expresia '3<x and x<=7' este adevărată dacă x este:
   a. 3,4,5,6 or 7
   b. 2,3,4,5 or 6
   c. 4,5, or 6
   d. 4,5,6 or 7

3. În Python, care expresie ar returna un rezultat diferit față de celalte 3?
   a. a < b < c
   b. c > b > a
   c. a < b or b < c
   d. a < b and b < c

CAPITOLUL 8 –
TIPURI DE DATE AGREGAT

La finalul acestui capitol, veți putea să:

- Înțelegeți tipurile de date agregat
- Utilizați în cadrul unui program date tip agregat: listă și tuplu

8.1 TIPURI DE DATE AGREGAT ÎN PYTHON

Concepte

Pe lângă tipurile standard de date precum integer, string și Boolean, există o altă categorie de tipuri de date denumite agregat. Acestea stochează elemente multiple, ca de exemplu o listă de nume. Majoritatea limbajelor de programare abordează într-un anumit fel tipurile de date agregat, ca de exemplu un tablou ce conține un grup de elemente din același tip de date

În Python există 2 tipuri de date agregat: **listă** și **tuplu**.

Listă

Reprezintă o colecție de valori. Listele pot fi modificate prin adăugarea sau eliminarea elementelor din conținutul lisei sau prin modificarea elementelor existente. Python poate afișa o listă întreagă de valori. De asemenea, el poate prelua elemente dintr-o listă. (În Python, tipul de date agregat 'listă' îndeplinește rolul de 'tablou (array)' existent în alte limbaje de programare.)

Tuplu

Un tuplu este exact ca o listă, cu precizarea că, odată creat, nu poate fi modificat prin rearanjarea, adăugarea, eliminarea sau modificarea elementelor individuale.

Chiar dacă restricția privind tuplurile pare să le facă mai puțin folositoare decât listele, tuplurile au două avantaje importante:

1. Preluarea elementelor din tupluri este mai rapidă decât cea din liste.
2. Un programator care citește codul pentru tupluri nu trebuie să verifice dacă valorile au fost modificate.

Itemii dintr-o listă sau dintr-un tuplu poartă numele de **elemente**.

Un element dintr-o listă sau dintr-un tuplu poate fi selectat prin stabilirea numelui listei sau tuplului și apoi a numărului elementului dorit în cadrul listei, scris între paranteze drepte. De exemplu, elementul 3 al listei numite pets este:

pets[3]

Elementele sunt menționate prin numere pornind de la 0. Astfel, în lista:

["pisică","câine","hamster","peștișor","iepure","broască"]

pets[0] = pisică

pets[1] = câine

pets[3] = peștișor

8.2 LISTE

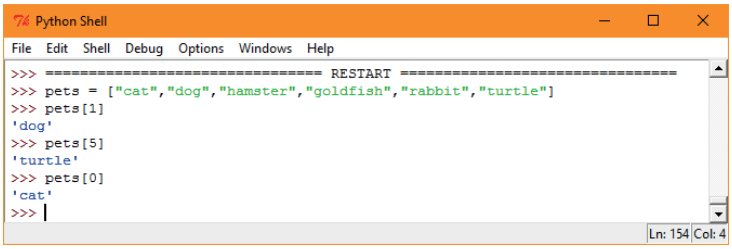
Concepte

Python conține anumite metode predefinite pentru lucrul cu listele. Spre exemplu, această comandă va sorta lista numită pets:

pets.sort()

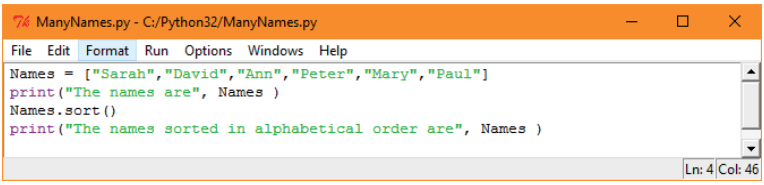
Exemplu: Lucrul cu liste

1. Deschideți Python.
2. Creați o listă numită pets, incluzând elementele între paranteze drepte, așa cum observați în imaginea de mai jos.
3. Explorați elementele individuale ale listei utilizând numere incluse între paranteze drepte, la fel ca mai jos.

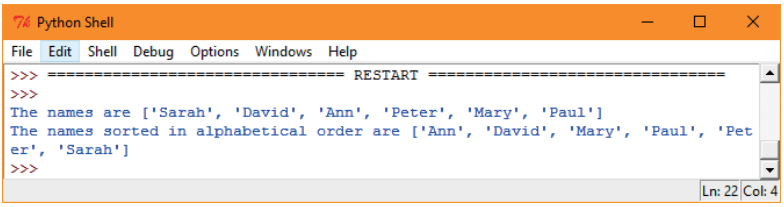


Exemplu: Sortarea unei liste în Python

1. Deschideți Python.
2. Creați un fișier numit ManyNames.py.
3. Tastați în program codul de mai jos, utilizând numele unor prieteni reali de ai dvs. în locul celor menționați mai jos.



4. Rulați programul.



8.3 TUPLURI

Concepte

Tupluri în Python

Tuplurile sunt create în același fel ca și listele, însă utilizând paranteze rotunde în loc de cele drepte.

Un tuplu nu poate fi sortat folosind comanda **sort()**, întrucât Python nu permite rearanjarea elementelor unui tuplu.

Chiar dacă o variabilă este atribuită unui tuplu, Python tot nu poate modifica acel tuplu. Python poate totuși să atribuie o nouă variabilă tuplului respectiv, înlocuind astfel tuplulul cu unul nou. Secvența de mai jos prezintă un exemplu al modului în care se poate înlocui un tuplu cu altul:

animal_tuple = (‘pisică’,’câine’,’porc’)
animal_tuple = (‘elefant’, ‘tigru’, ‘girafă’)

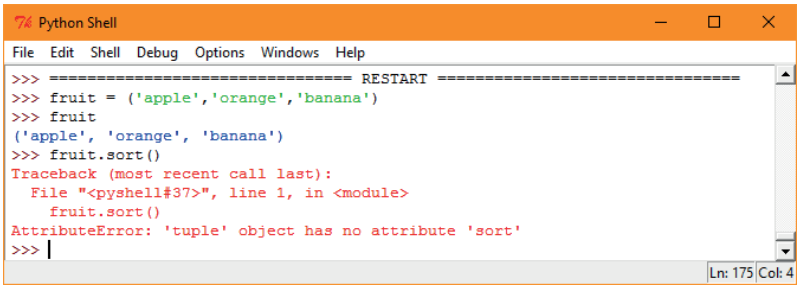
Cel de al doilea tuplu îl înlocuiește pe primul.

O utilizare comună a unui tuplu o reprezintă stocarea poziției pe coordonatele x și y a unui punct pe ecran. În loc să existe 2 variabile, una pentru x și alta pentru y, o singură variabilă tuplu poate stoca ambele valori. Când sunt necesare valorile pentru x și y, acestea pot fi extrase din tuplu.

(O listă poate fi utilizată în același mod ca și un tuplu pentru stocarea poziției pe coordonatele x și y a unui punct pe ecran, însă un tuplu este mai eficient.)

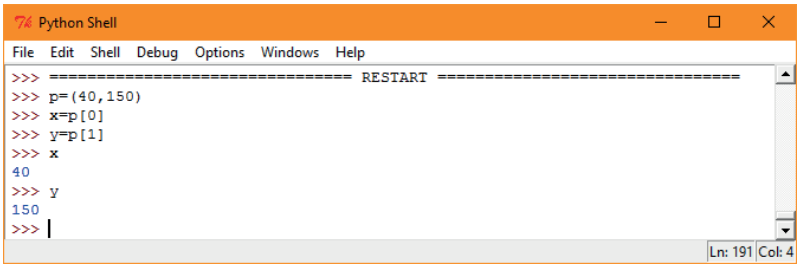
Exemplu: Încercarea de a sorta un tuplu

1. Deschideți Python.
2. Utilizați codul de mai jos pentru a crea un tuplu ce conține nume de fructe.
3. Afișați valorile tuplului tastând 'fruit'.
4. Încercați să sortați tuplul.



Exemplu: Extragerea valorilor x și y din tuplu

1. Deschideți Python.
2. Utilizați codul de mai jos pentru a crea un tuplu ce conține valorile 40 și 150.
3. Extrageți valorile x și y din tuplu.
4. Afișați valorile x și y.



8.4 EXERCIȚII RECAPITULATIVE

1. O variabilă de tip listă poate stoca:
    a. Doar un singur număr între -128 și 128.
    b. Un fișier video într-un format uzual pentru afișarea pe ecran.
    c. Un fișier audio într-un format uzual pentru tonurile de apel la telefoanele mobile.
    d. O listă de studenți ce poate fi sortată utilizând funcția sort().

2. Ce tip de date reprezintă în Python fiecare din valorile de mai jos?
    a. 37
    b. ‘Sarah’
    c. (100, 20, 25)
    d. [100, 20, 25]
    e. 15.9
    f. 999999999999999999999999999999999
    g. -37.0
    h. [‘David’, ‘Sarah’, ‘Ann’, ‘Bill’]
    i. [‘Red’, ‘Green’, ‘Blue’]
    j. ‘[100, 20, 25]’

CAPITOLUL 9 –
CONSTRUIREA CODURILOR

La finalul acestui capitol, veți putea să:

- Descrieți caracteristicile unui cod bine structurat și documentat: indentare, comentarii adecvate, utilizarea numelor descriptive
- Definiți termenul de comentariu și să identificați scopului unui comentariu într-un program
- Utilizați comentarii în cadrul unui program

9.1 COD

Concepte

Imaginați-vă că tocmai ați finalizat un program pentru un joc pe care l-ați inventat. Cunoașteți exact ce face codul din program întrucât dvs. l-ați scris. Vă salvați fișierele create și treceți la urmatorul proiect. Imaginați-vă apoi că reveniți la acest program după câteva luni; cât de repede v-ați putea familiariza cu codul respectiv? Cât de ușor ar fi pentru o persoană care nu a văzut niciodată codul respectiv să îl înțeleagă?

În realitate, programele create în Python pot fi destul de mari și pot conține multe instrucțiuni. De obicei, la aceste programe lucrează mai mulți programatori, în diverse etape de dezoltare a programului. Este esențial ca programul să conțină un cod care să poată fi 'inteligibil' adică persoana care îl citește să poate înțelege ușor ce anume face programul și de ce.

Există o serie de tehnici generale de a face un cod mai ușor de înțeles, care se pot aplica în toate limbajele de programare, și anume:

- Utilizarea comentariilor
- Organizarea codului
- Utilizarea numelor descriptive

9.2 COMENTARII

Concepte

Un **comentariu** reprezintă o porțiune de text care explică ce anume face respectiva porțiune de cod. Este o descriere scurtă a unei porțiuni de cod pe care oamenii o pot citi, însă pe care computerele o vor ignora, fiind proiectată pentru a-i ajuta pe programatori să înțeleagă ce anume se întâmplă în cadrul programului. Comentariile adecvate reprezintă o caracteristică a unui cod bine structurat și documentat. Comentariile din cadrul codului ar trebui să îl ajute pe autor, dar și pe alți oameni să înțeleagă ce anume face fiecare secțiune de cod.

Comentariile încep cu simbolul # și apar scrise cu roșu în editorul Python.

Textul roșu care apare între semnul # și sfârșitul rândului nu este luat în considerare de Python la rularea codului. Acest text este doar pentru programatori.

```
# Acesta este un comentariu
```

9.3 ORGANIZAREA CODULUI

Concepte

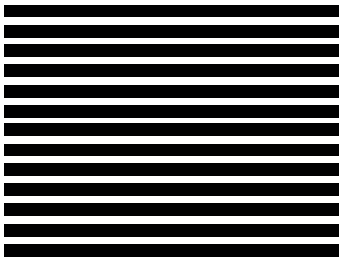
Programatorii fac codul mai ușor de urmărit prin divizarea lui în porțiuni sau blocuri mai mici.

Un bloc este definit ca un grup adiacent de linii care sunt indentate la fel. Indentarea este utilizată pentru a putea înțelege și citi mai ușor codul respectiv. Mulți editori de coduri realizează indentare automată.

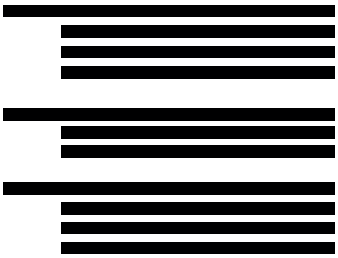
Blocurile sunt de obicei denumite folosindu-se cuvântul 'def' urmat de nume, ales pentru a descrie ce anume face codul respectiv. (Aceste aspecte sunt detaliate în capitolul 11 Proceduri și Funcții.)

Numele ("def name") este introdus primul, iar apoi începutul blocului este evidențiat printr-o indentare a liniei. Liniile de cod din cadrul unui bloc au aceeași indentare și reprezintă o secvență de instrucțiuni și, ca urmare, vor fi rulate unele după celelaltele.

Prin utilizarea unor blocuri denumite corect, un programator poate oferi o structură clară a unui program Python destul de complicat.



Cod simplu, neseparat pe blocuri



Cod organizat pe blocuri

9.4 NUME DESCRIPTIVE

Concepte

Numele descriptive reprezintă o caracteristică a unui cod bine structurat și documentat. În programare, reprezintă procesul de a atribui nume semnificative elementelor, funcțiilor și procedurilor. Există o structură acceptată, comună tuturor limbajelor de programare. Spre exemplu, pentru a scrie în Python o mică funcție numită 'egg timer', aceasta ar trebui denumită 'egg_timer'. Acest lucru permite utilizatorilor să înțeleagă scopul funcției respective.

Atribuirea unor nume semnificative variabilelor va oferi mai mult sens utilizatorului. Numele variabilelor ar trebui alese cu grijă pentru a oferi indicii despre scopul utilizării variabilei respective.

În lista noastră de exemple de la capitolul 8, variabila numită **pets** ar fi putut fi utilizată în schimb pentru o listă de fructe:

pets=['ananas','mango','kiwi']

Python nu ar avea nicio problemă și comanda:

print(pets[1])

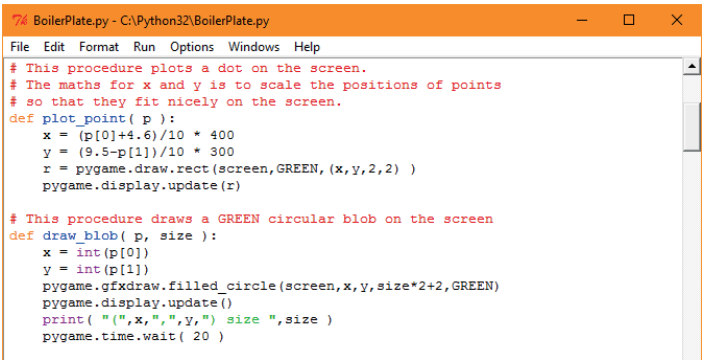
ar afișa:

'mango'

Cu toate acestea, utilizarea unui astfel de nume va crea confuzie pentru un programator care încearcă să citească codul respectiv.

Exemplu: Tranformarea unui cod complex într-unul mai ușor de citit

1. Studiați programul Python prezentat mai jos. Este o porțiune complexă de cod, utilizată aici pentru a ilustra unele din tehnicile discutate mai sus pentru îmbunătățirea lizibilității codului.
2. Încercați să identificați în cod următoarele:
   a) Comentariile care să explice funcționarea programului
   b) Numele blocurilor de cod



3. Revizuiți codurile create până acum pe baza acestui material și verificați unde anume ați putea îmbunătăți lizibilitatea codului pe baza tehnicilor prezentate în acest capitol.

9.5 EXERCIȚII RECAPITULATIVE

1. Un 'comentariu' în cadrul unui program reprezintă:
   a. Orice este inclus între ghilimele
   b. Orice este inclus în cod, înaintea unei proceduri
   c. Caracterul ASCII '#'
   d. Un text care ajută la documentarea programului.

2. Ar trebui să utilizați un comentariu:
   a. După fiecare linie din program.
   b. Înaintea fiecărei linii din program.
   c. Pentru a ajuta o persoană să înțeleagă programul creat de dvs.
   d. Numai pentru funcția cea mai dificilă din program.

3. Numele variabilelor ar trebui să fie:
   a. Scurte pentru a face programul să ruleze cât mai rapid posibil.
   b. Format din minim 8 litere astfel încât să nu fie ușor de ghicit de alte persoane.
   c. Format din litere și cifre, nu numai din litere.
   d. Alese astfel încât să facă programul mai ușor de înțeles.

4. Blocurile de cod sunt indicate astfel:
   a. Prin încadrare între ghilimele.
   b. Încep cu '{' și se termină cu '}'.
   c. Încep cu '(' se termină cu ')'.
   d. Prin indentarea blocurilor de cod.

CAPITOLUL 10 –
INSTRUCȚIUNI CONDIȚIONALE

La finalul acestui capitol, veți putea să:

- Definiți termenul de instrucțiune condițională şi să identificați scopul acesteia în cadrul unui program.
- Definiți termenul de test logic şi să identificați scopul acestuia în cadrul unui program
- Utilizați expresiile logice Booleene în cadrul unui program
- Utilizați instrucțiunile condiționale IF…THEN...ELSE într-un program

10.1 SECVENȚE ȘI INSTRUCȚIUNI

Concepte

În limbajele de programare mai vechi, fiecare linie de cod dintr-un program conținea o instrucțiune care se executa singură. Fiecare instrucțiune era executată în ordine de computer, una după cealaltă. Acest lucru poartă denumirea de **programare secvențială.**

În **programarea secvențială** instrucțiunile sunt executate în ordine, una după cealaltă.

În limbajele de programare, o **instrucțiune** reprezintă cea mai mică porțiune de cod care se poate executa singură.

Un exemplu de instrucțiune este print():

```
print( 'Ceva de afișat')
```

Atunci când programatorii utilizează termenul de 'instrucțiune', ei fac referire la mici secțiuni de cod precum print().

10.2 INSTRUCȚIUNEA IF

Concepte

În cadrul unei scheme logice, un bloc de decizie poate fi considerat o element complet autonom. Acest bloc de decizie are un rezultat Adevărat sau Fals care permite luarea unei decizii cu privire la ce anume ar trebui codul să execute în continuare.

O **instrucțiune condițională** este utilizată pentru a evalua o expresie ca fiind Adevărată sau Falsă. Rezultatul, respectiv valoarea de True sau False, determină pașii următori.

În Python, instrucțiunile condiționale utilizează un format și un aspect structurat și sunt scrise folosind cuvântul cheie **if** urmat de o expresie booleană ce evaluează condiția și apoi semnul :

De exemplu:

```
Age = input ("Ce vârstă ai?")

if (int (Age ) < 14 ) :

print ( "Esti mai tanar/a ca mine" )
```

Blocul de cod indentat din linia următoare (comanda print() în acest caz) va fi executat dacă expresia va fi evaluată ca fiind adevărată.

Expresia booleană (int (Age) < 14) din cadrul instrucțiunii condiționale if poartă denumirea de **test logic**.

Un **test logic** reprezintă o expresie care oferă un răspuns de tip Yes/No, True/ False. Răspunsul afectează rularea codului în continuare.

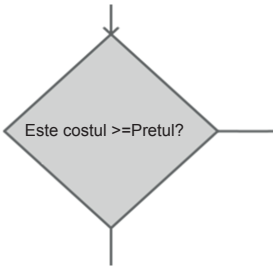
Acesta este formatul structurat care trebuie folosit:

If expression
statement if true

În pseudocod testul logic poate fi exprimat în limbaj natural astfel:

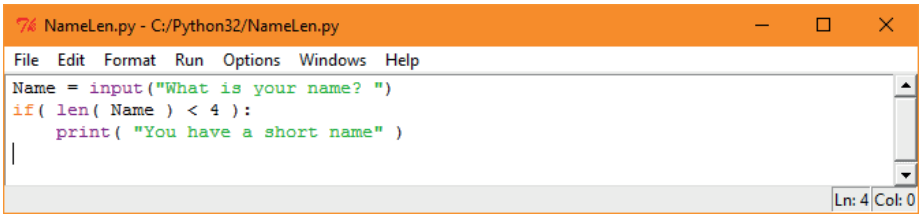
'Ai suficienți bani?'

Același test logic ar putea fi prezentat sub forma unei scheme logice astfel:

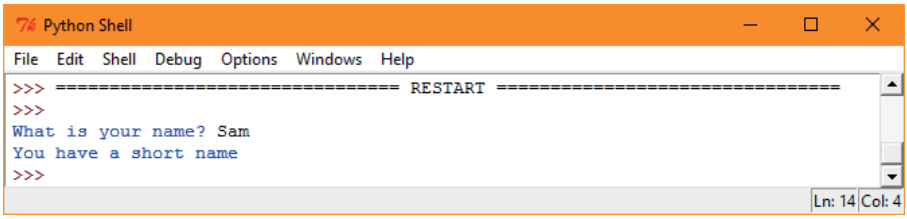


Exemplu: Un test logic

1. Deschideți Python.
2. Creați un nou fișier numit NameLen.py. Scrieți codul de mai jos. Asigurați-vă că utilizați simbolul : la finalul liniei care începe cu cuvântul cheie if.



3. Rulați programul.
4. Răspundeți la întrebare prin introducerea fie a unui nume scurt, fie a unuia lung (mai puțin sau mai mult de 4 litere).
5. Observați rezultatul.



10.3 INSTRUCȚIUNEA IF…ELSE

Concepte

Python poate permite unui bloc de cod indentat, unei subrutine, să ruleze dacă testul logic este adevărat și unui alt bloc de cod să ruleze dacă testul logic este Fals.

Codul este scris tot folosind instrucțiunea **if**, ca și până acum, la care se adaugă cuvântul cheie **else**, urmat de simbolul : pentru a adăuga codul ce va rula în cazul în care testul logic este fals.

Acesta este formatul structurat care trebuie folosit:

```
if expression:
        statement if true
else:
        statement if false
```

De exemplu:

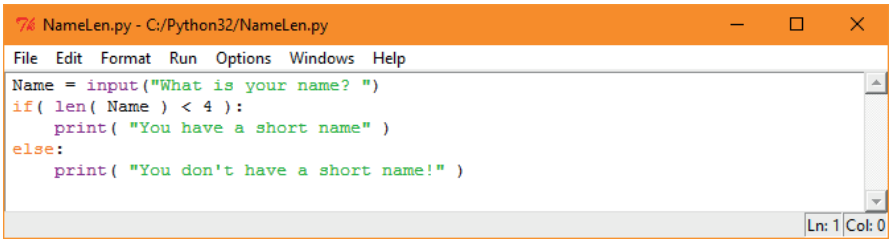
```
Age = input ("Ce vârstă ai?")

if (int (Age ) < 14 ) :

    print ( " Esti mai tânăr/ă ca mine " )

else:

    print ( " Esti mai bătrân/ă ca mine " )
```

Blocul de cod indentat din linia următoare (comanda print() în acest caz) va fi executat dacă expresia va fi evaluată ca fiind adevărată.

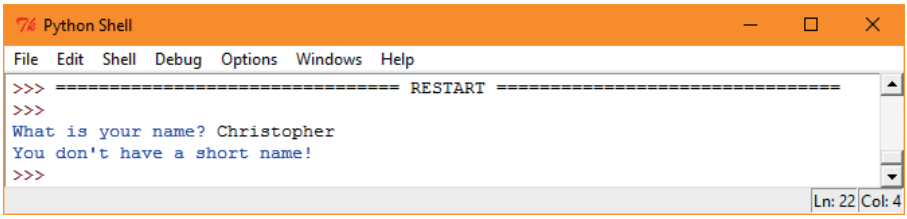
Notă: Comanda 'else' nu este indentată, ea având același nivel de indentare ca și cuvântul cheie 'if'. Python inițial va indenta cuvântul cheie 'else', așa că va trebui să apăsați tasta Backspace pentru a obține nivelul corect de indentare.

Exemplu: O instrucțiune if..else.

1. Deschideți exemplul NameLen.py.
2. Modificați programul conform imaginii de mai jos.
3. Atunci când ajungeți la momentul scrierii comenzii 'else' apăsați tasta Backspace pentru a anula un nivel de indentare.
4. Nu uitați să puneți simbolul : după cuvântul cheie else.



5. Rulați codul.
6. Introduceți un nume scurt.
7. Rulați codul din nou.
8. Introduceți un nume mai lung pentru a verifica executarea codului în cazul în care testul logic este fals.



10.4 EXERCIȚII RECAPITULATIVE

1. O instrucțiune condițională Python:
    a. Conține cuvântul cheie ‘if’
    b. Conține cuvântul cheie ‘def’
    c. Este un tip special de comentariu care precizează în ce condiții ar trebui să fie utilizat codul.
    d. Este un tip special de procedură ce poate fi o funcție în anumite condiții.

2. În următorul cod:

```
if password == ‘Voldemort’ :
        print( ‘I guessed your password correctly’ )
else :
        print( ‘I did not guess your password correctly’ )
```

…care dintre următoarele afirmații este adevărată?
    a. Atât primul, cât și cel de al doilea bloc de cod indentat vor fi rulate întotdeauna.
    b. Nici primul bloc de cod indentat, nici al doilea nu vor fi rulate niciodată
    c. Este o eroare logică întrucât nu se poate folosi == cu un șir de caractere.
    d. Primul sau al doilea bloc de cod indentat ar putea fi rulat, în funcție de valoarea parolei.

CAPITOLUL 11 –
PROCEDURI ȘI FUNCȚII

La finalul acestui capitol, veți putea să:

- Înțelegeți termenul de parametru și să identificați scopul utilizării acestuia în cadrul unui program.
- Înțelegeți termenul de procedură și să identificați scopul utilizării acesteia în cadrul unui program
- Scrieți și să definiți o procedură în cadrul unui program
- Înțelegeți termenul de funcție și să identificați scopul utilizării acesteia în cadrul unui program
- Scrieți și să definiți o funcție în cadrul unui program

11.1 SUBRUTINE

Concepte

O porțiune de cod poate fi utilizată de mai multe ori în diverse secțiuni ale unui program. Aceasta poartă numele de subrutină.

O **subrutină** este un bloc de cod care poate fi apelat și rulat de mai multe ori în cadrul unui program. Spre exemplu, o rutină ar putea fi utilizată pentru afișarea datei sau orei pe ecranul monitorului.

În Python cuvântul cheie ‘def’, (prescurtarea de la 'define') este utilizat pentru a defini un nume pentru o subrutină. Putem apoi **apela** sau **invoca** o subrutină de oriunde din întregul program prin referirea la numele ei. Rubrutina va rula și când se va finaliza, programul principal va fi reluat din punctul aflat înainte de începerea subrutinei.

O subrutină poate efectua un calcul și returna un răspuns sau o valoare. Acest lucru se numește **returnarea** unei valori.

Există 2 tipuri de subrutine.

- **Funcții**

O funcție reprezintă o subrutină ce calculează o valoare în cadrul unui program.

- **Proceduri**

O procedură reprezintă o subrutină ce execută o acțiune, fără însă a returna o valoare.

11.2 FUNCȚII ȘI PROCEDURI

Concepte

Funcțiile și procedurile pot deveni mai flexibile prin intermediul **parametrilor**.

Un **parametru** reprezintă o variabilă specială ce este utilizată într-o subrutină pentru a influența modul de funcționare a acesteia.

Exemplu: Apelarea funcțiilor și procedurilor predefinite

Python conține anumite subrutine (funcții și proceduri) predefinite.

input() și print() sunt amândouă subrutine:

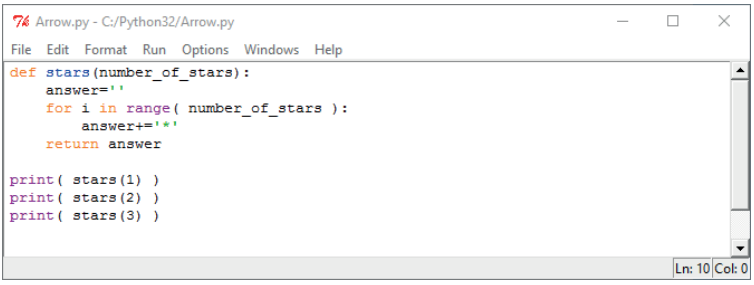
- input() este un exemplu de funcție. Când este apelată în cadrul unui program, execută o acțiune și returnează o valoare înapoi codului care a apelat-o. Programul se reia apoi din punctul în care a fost apelată funcția input() în cadrul programului.
- print() este un exemplu de procedură. Când este apelată în cadrul unui program, execută o acțiune și apoi programul se reia din punctul în care procedura print() a fost apelată.

Exemplu: Crearea unei funcții

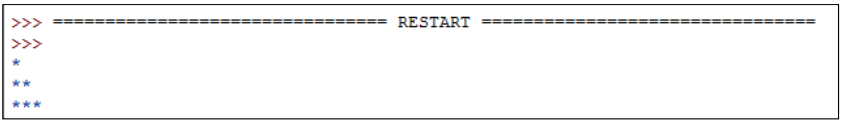
Această porțiune de cod definește o funcție numită ‘stars’ ce conține un **parametru** numit ‘number_of_stars’.Subrutina va crea un șir de caractere ‘*’ având lungimea dată de parametrul number_of_stars și va afișa pe ecran șirul creat.

Există 3 instrucțiuni de afișare în program care trebuie să verificăm dacă funcționează corect.

1. Deschideți Python.
2. Creați un nou fișier numit Arrow.py.
3. Introduceți primul bloc de cod din imaginea de mai jos, cu instrucțiunea def și toate liniile de cod indentate, pentru a crea o nouă funcție.
4. Atunci când introduceți cele 3 instrucțiuni de afișare, apelați funcția de 3 ori.



5. Rulați programul pentru a vedea rezultatul.

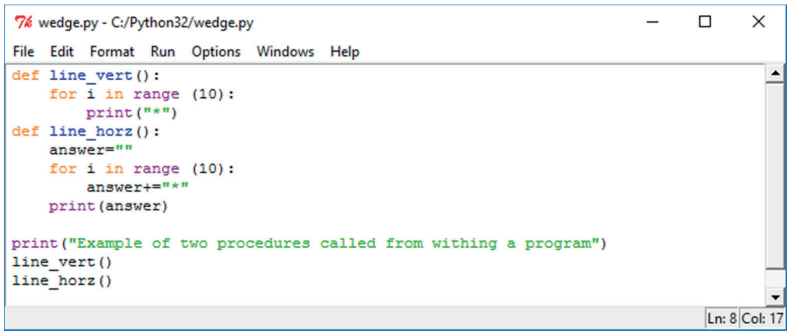


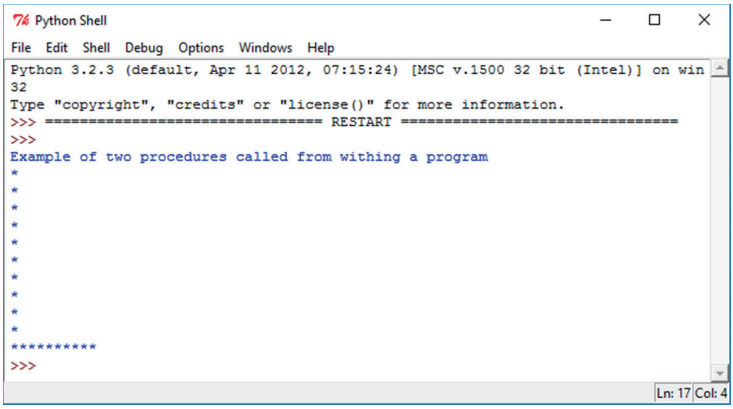
Notă: Programul rulează în ordine de la prima instrucțiune de afișare.
Prima instrucțiune print() va afișa o stea pe prima linie.
A doua instrucțiune print() va afișa 2 stele pe a doua linie.
A treia instrucțiune print() va afișa 3 stele pe a treia linie.

Cantitatea de stele afișată pe fiecare rând depinde de valoarea parametrului number_of_stars din fiecare instrucțiune print().

Exemplu: Crearea Procedurilor

1. Deschideți Python.
2. Creați un nou fișier numit Lines.py.
3. Introduceți cele două blocuri de cod din imaginea de mai jos, cu cele 2 instrucțiuni def și toate liniile de cod indentate, pentru a crea 2 noi proceduri.
4. Apoi tastați instrucțiunea de afișare și cele 2 linii ce invocă cele 2 proceduri din corpul principal al programului.
5. Salvați programul și rulați-l pentru a vedea rezultatul.





Cum funcționează programul:

- Programul rulează în ordine de la prima instrucțiune de afișare
- Programul apelează procedura line_vert ce va afișa o linie verticală de stele pe ecran. După executarea acestei proceduri, programul se reia cu următoarea instrucțiune care apelează procedura numită line_horz.
- Ambele proceduri sunt rulate, însă ele nu returnează nicio valoare programului.

11.3 EXERCIȚII RECAPITULATIVE

1. O funcție tipică în Python:
   a. Începe cu cuvântul cheie def și oferă o porțiune de cod care returnează o valoare.
   b. Începe cu cuvântul cheie abc și oferă o porțiune de cod care returnează o valoare.
   c. Desemnează o porțiune de cod care execută o serie de instrucțiuni și apoi oprește programul.
   d. Desemnează o porțiune de cod care execută o serie de instrucțiuni și apoi repornește interpretorul Python shell.

2. O procedură tipică în Python:
   a. Constă în pseudocod pe care computerul nu trebuie să îl ruleze.
   b. Constă în totalitate din comentarii care specifică scopul programului.
   c. Începe cu #
   d. Este exact ca o funcție, însă nu returnează o valoare.

3. Un parametru este:
    a. O valoare care este transmisă unei proceduri sau unei funcții în scopul folosirii acesteia
    b. O metodă de măsurare aproximativă a distanței, ce ar putea simplifica anumite porțiuni de cod.
    c. O modalitate prin care un program poate executa mai multe acțiuni în același timp.
    d. Un bloc de cod ce conține un anumit număr de linii.

CAPITOLUL 12 –
BUCLE (LOOP)

La finalul acestui capitol, veți putea să:

- Înțelegeți termenul de buclă și să identificați scopul utilizării acestora în cadrul unui program
- Recunoașteți tipurile de bucle utilizate în cadrul iterațiilor: for, while, repeat
- Înțelegeți termenul de buclă infinită
- Utilizați iterațiile în cadrul unui program: for, while, repeat

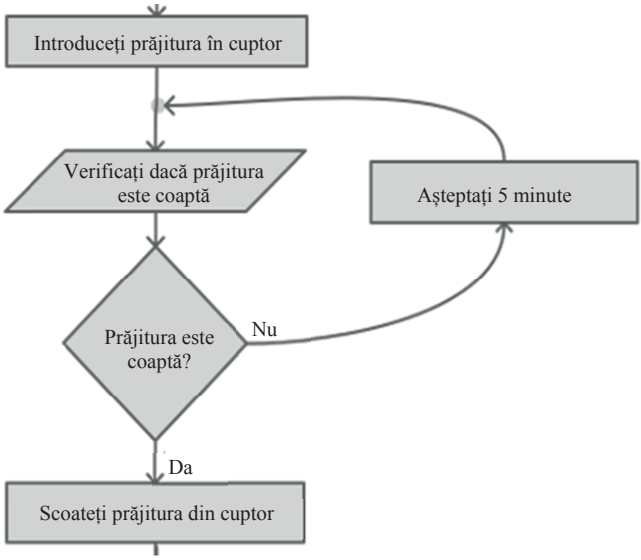
12.1 BUCLE (LOOP)

Concepte

Până în acest punct, am creat programe care funcționează secvențial, adică instrucțiunile sunt executate în ordinea în care sunt scrise, similar cu respectarea unei rețete pas cu pas. Dacă este necesar să se execute același pas de mai multe ori, putem utiliza ceea ce în programare poartă numele de **buclă (loop)**.

O buclă (**loop**) reprezintă o porțiune de cod care este rulată în mod repetat în anumite condiții.

O buclă este reprezentată în cadrul unei scheme logice printr-o săgeată care pornește de la un bloc de decizie și este orientată înpoi spre blocul anterior din secvența de instrucțiuni. Examinați schema logică de mai jos pentru coacerea unei prăjituri. Verificarea prăjiturii pentru a vedea dacă este sau nu coaptă poate fi repetată de mai multe ori înainte ca prăjitura să fie gata pentru a fi scoasă din cuptor. Acest pas repetitiv este reprezentat printr-o buclă – verificați prăjitura și dacă nu este coaptă, așteptați 5 minute și verificați încă o dată.



O buclă în cadrul unei scheme logice

Utilizarea unei bucle pentru repetarea unei acțiuni reprezintă una dintre cele mai utile tehnici în programare. Codul pentru buclă (loop) este prezent în majoritatea limbajelor de programare, cu mici variații legate de convențiile de nume e.g. 'repeat' este utilizat pentru a determina numărul de repetări în anumite limbaje de programare; vom vedea mai târziu în acest capitol că 'while' face acest lucru în Python.

Python folosește instrucțiunea '**for'** pentru a crea bucle.

Instrucțiunea **for** conține mai multe părți.

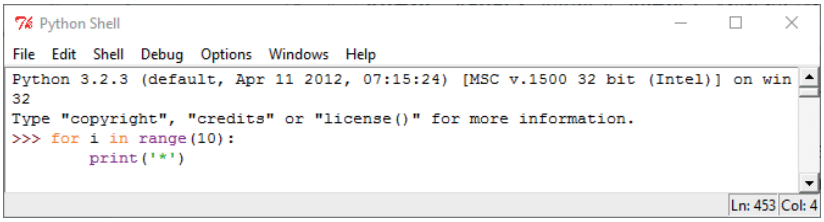
```
for i in range( 10 ):
        secvență_de_instrucțiuni
```

- Cuvântul cheie 'for' este urmat de numele variabilei, în acest caz i, ținând cont de repetițiile buclei. Variabila i va avea o valoare inițială 0 și, de fiecare dată când bucla va fi rulată, va crește la 1, apoi la 2 și tot așa până la 9.
- Cuvintele 'in range()' cu un număr între paranteze, urmat de simbolul : indică numărul de rulări ale buclei.
- O linie de cod indentat conținând o instrucțiune va fi executată de 10 ori.

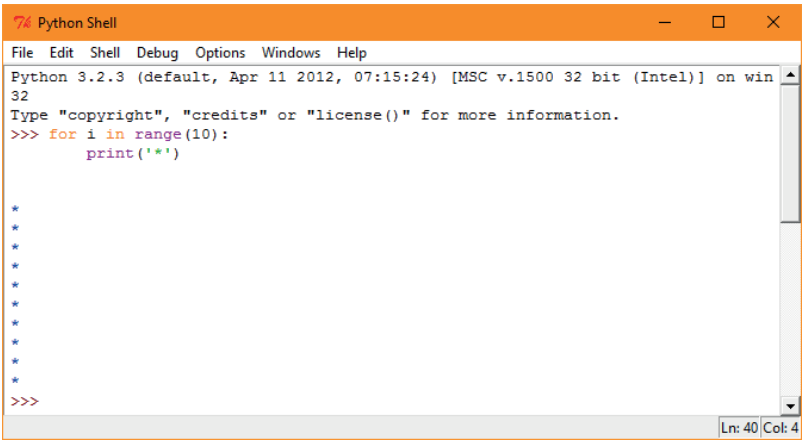
Exemplu: Încercarea de a afișa un rând cu 10 stele

Acest exemplu prezintă o încercare de a afișa un rând cu 10 stele.

1. Deschideți Python.
2. Introduceți următorul cod.



3. Apăsați Enter pentru a rula codul.



12.2 BUCLE FOLOSIND VARIABILE

Concepte

În exemplul anterior, fiecare stea era afișată pe un rând separat. Se dorea un rând de stele. O altă abordare ar fi:

```
print('**********')
```

Această abordare va funcționa, însă nu este eficientă – dacă vreau să afișez 599 de stele?

În acest caz, utilizarea buclelor economisește foarte mult timp și scurtează codul. Putem scrie o buclă care să afișeze 10 stele, 42 de stele sau 599 de stele, în funcție de o valoare.

Cum funcționează programul:

- Pentru a afișa un rând de 10 stele prin utilizarea unei bucle, trebuie să construiți rândul de stele într-o variabilă. Apoi, odată ce rândul este gata, codul îl poate afișa. Această abordare va utiliza operatorul + pentru a adăuga șirul '*' la variabila numită answer.

answer = answer + '*'

Această linie de cod utilizează valoarea curentă a variabilei answer, adaugă o stea la final și salvează rezultatul obținut înapoi în variabila answer.

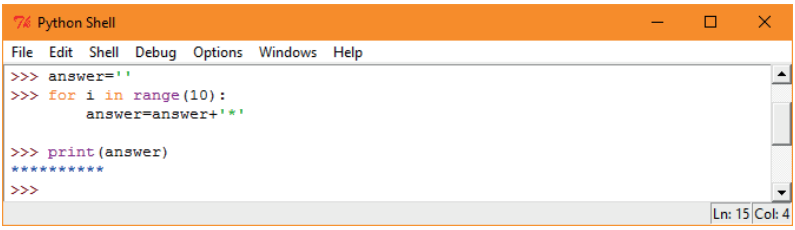
- Variabila answer trebuie întâi să fie inițializată, înainte de a fi utilizată. Pentru a o inițializa cu un șir de 3 stele utilizați codul:

answer = '***'

Pentru a inițializa variabila answer într-un șir gol, ștergeți cele 3 stele din linia de mai sus. Păstrați cele 2 ghilimele! Fără ele, programul va da eroare. Cele 2 ghilimele de la început și de la final îi spun programului Python că variabila answer stochează un șir de caractere.

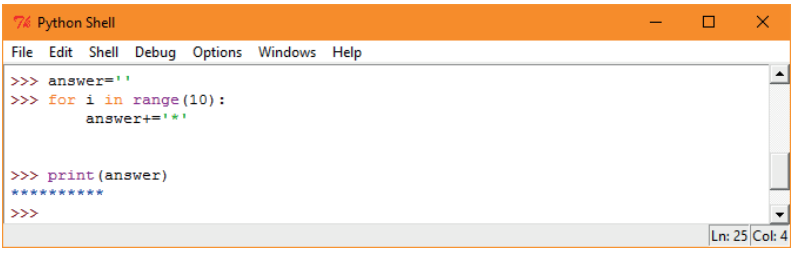
Exemplu: Rând de stele utilizând o variabilă

1. Deschideți Python.
2. Inițializați variabila numită answer ce conține un șir de caractere gol.
3. Utilizați o buclă pentru a adăuga 10 stele în șir.
4. Afișați șirul de stele.



Există o metodă mai scurtă de scriere answer=answer+'*' care utilizează noul operator +=.

5. Rescrieți codul utilizând de data aceasta operatorul +=, așa cum este prezentat mai jos:



12.3 TIPURI DE BUCLE

Concepte

Am văzut că buclele au o structură și un aspect particular în Python.

```
for i in range( 10 ):
    secvență_de_instrucțiuni
```

Blocul de cod indentat, după linia ‘for’, se consideră că se află 'în interiorul buclei'.

- Codul aflat înainte de buclă se execută și apoi se execută bucla.
- Codul aflat în interiorul buclei este executat de mai multe ori.
- Codul aflat imediat după buclă este executat doar după ce bucla a finalizat de repetat codul aflat în blocul indentat.

Până acum am studiat bucla de bază, însă există câteva variații de bucle care se pot utiliza de asemenea.

Un alt tip de buclă utilizează un test logic în loc de range(). Aceasta poartă numele de buclă while.

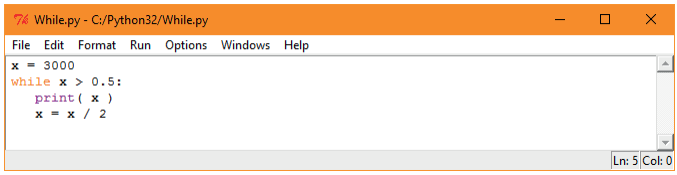
O **buclă while** testează o expresie booleană și repetă execuția buclei atâta timp cât (i.e. While) expresia booleană este adevărată.

Cum funcționează programul:

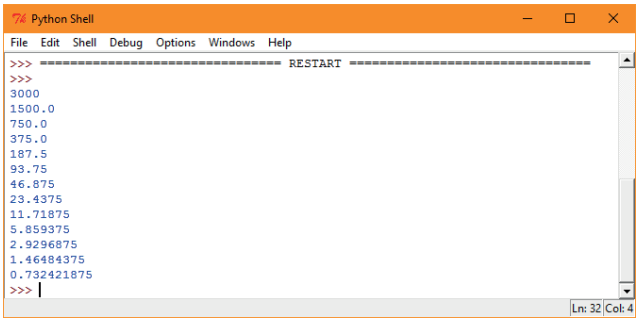
- Exemplul următor va inițializa o variabilă x la 3000 și o va împărți la 2 atâta timp cât x este mai mare ca 0.5.
- Apoi se va opri.

Exemplu: O buclă While

1. Deschideți Python.
2. Creați un nou fișier numit While.py.
3. Tastați codul de mai jos:



4. Rulați programul.



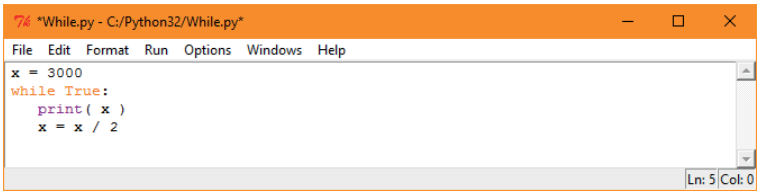
Într-o buclă While, dacă testul logic este mereu adevărat, bucla se va repeta la infinit. Ca urmare, va deveni o buclă infinită.

O buclă infinită nu se oprește niciodată. Ea folosește cuvintele cheie "while true", urmate de semnul :

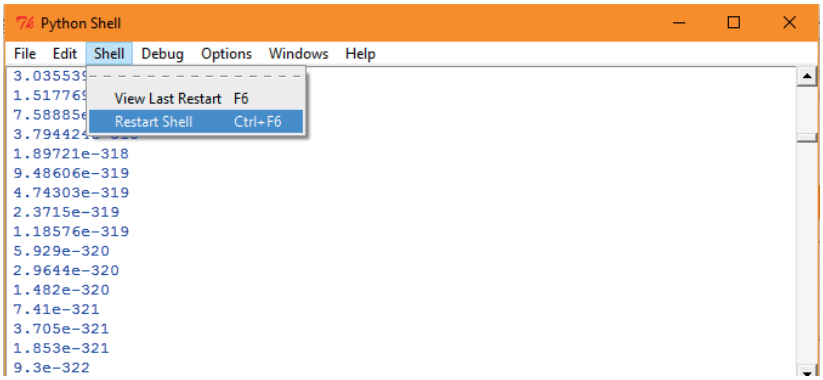
Întrucât ne dorim de obicei un program care să se oprească la un moment dat, buclele infinite indică de obicei o eroare.

Exemplu: O buclă infinită

1. Deschideți fișierul While.py.
2. Modificați testul logic astfel încât linia să conțină textul While True:



3. Rulați programul.
4. Puteți opri programul prin apelarea meniului Shell, comanda Restart Shell.

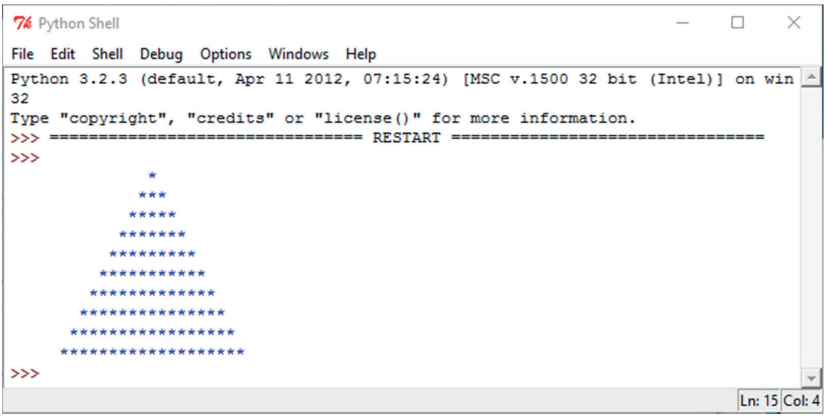


12.4 RECAPITULARE

Concepte

Exemplu: Desenarea unui vârf de săgeată

Python poate fi utilizat pentru a scrie un program ce desenează un vârf de săgeată, precum cel prezentat mai jos:

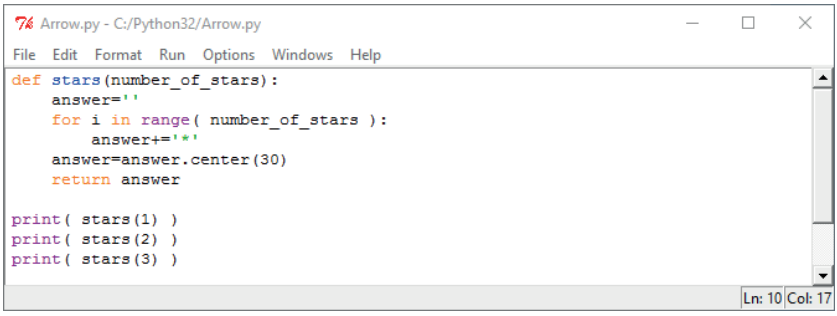


Acest lucru ar putea fi realizat utilizând multe instrucțiuni print(). Cu toate acestea, poate fi realizat cu mai puține instrucțiuni, prin utilizarea buclelor, funcțiilor și procedurilor.

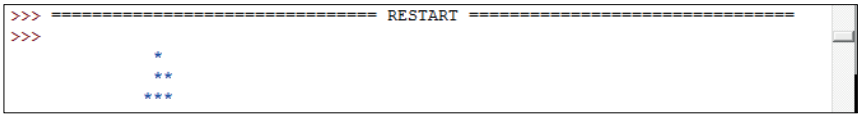
Pentru desenarea săgeții, fiecare rând de stele trebuie să fie centrat pe orizontală în loc să fie aliniat la stânga. Există o funcție care face acest lucru. Se numește center(). Dacă variabila numită answer stochează un șir de stele, următoarea linie de cod adaugă spații înainte și după stele pentru a crea 30 de caractere în total.

answer=answer.center(30)

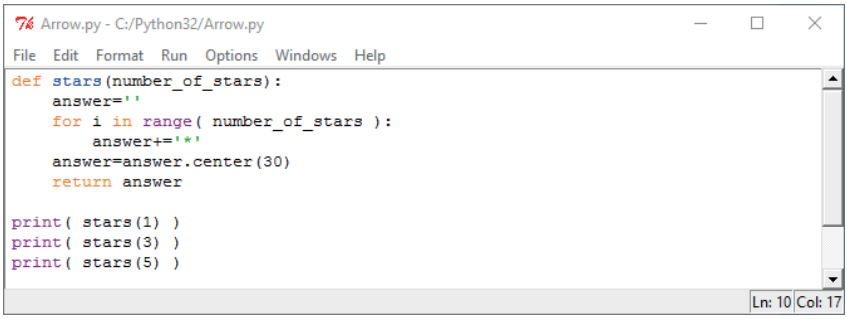
1. Deschideți fișierul Arrow.py creat la capitolul anterior.
2. Modificați codul după cum urmează pentru a centra stelele.



3. Rulați codul pentru a obține următorul rezultat.



4. Pentru a obține o formă similară unei săgeți, primul rând ar trebui să aibă o stea, al doilea 3 stele, al treilea 5 stele. Modificați valoarea parametrului din fiecare instrucțiune print (), după cum urmează, pentru a remedia acest lucru:



5. Rulați codul pentru a obține următorul rezultat:

```
>>> ================================ RESTART ================================
>>>
                              *
                             ***
                            *****
```

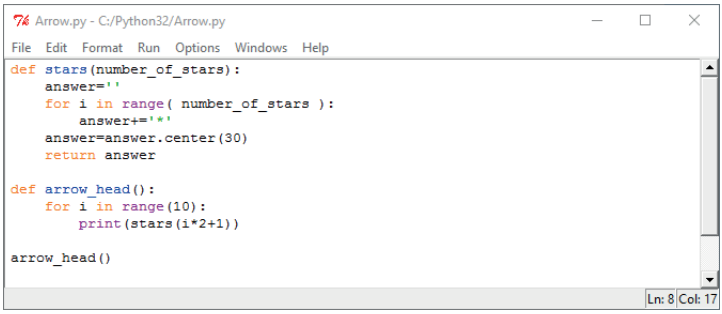
Putem construi o expresie și scrie o subrutină care să creeze vârful de săgeată și să reducă volumul de cod necesar. Avem nevoie de 10 rânduri de stele pentru a construi vârful de săgeată.

Luați în considerare expresia:

i*2+1

Atunci când i=0 valoarea expresiei este 1
Atunci când i=1 valoarea expresiei este 3
Atunci când i=2 valoarea expresiei este 5
Atunci când i=3 valoarea expresiei este 7
Atunci când i=4 valoarea expresiei este 9
Atunci când i=5 valoarea expresiei este 11
Atunci când i=6 valoarea expresiei este 13
Atunci când i=7 valoarea expresiei este 15
Atunci când i=8 valoarea expresiei este 17
Atunci când i=9 valoarea expresiei este 19

1. Atribuiți-i noii subrutine numele arrow_head().
2. Invocați arrow_head() ca ultima linie a programului.



Notă: Subrutina arrow_head() este o **procedură**, întrucât nu returnează nicio valoare.
Subrutina stars() este o **funcție**, pentru că returnează o valoare.

1. Verificați modul de funcționare al programului înainte de rularea codului:
   - Programul începe prin executarea procedurii arrow_head().
   - Variabila i va avea 10 valori de la 0 la 9 și ca urmare procedura arrow_head() va rula de 10 ori, de fiecare dată apelând funcția stars(). Acesta este un exemplu al unei subrutine care apelează o altă subrutină.
   - Funcția stars() conține o variabilă numită answer. Această variabilă va stoca un șir de stele pentru fiecare instrucțiune print ().
   - Subrutinele care apelează alte subrutine reprezintă una dintre cele mai importante modalități prin care problemele complexe sunt descompuse în probleme mai mici, fiecare părticică mai mică executând o parte mică, bine definită, a problemei globale.
2. Rulați fișierul Arrow.py pentru a vedea vârful de săgeată pe care codul o desenează.

```
>>> ================================ RESTART ================================
>>>
                *
               ***
              *****
             *******
            *********
           ***********
          *************
         ***************
        *****************
       *******************
      *********************
```

12.5 EXERCIȚII RECAPITULATIVE

1. Care este scopul principal al unei bucle?
   a. Pentru a preveni ieșirea din program la finalul acestuia.
   b. Pentru a vă deplasa înapoi la începutul programului.
   c. Pentru a conecta un computer la altul.
   d. Pentru a executa anumite instrucțiuni în mod repetat.

2. Cum se numește o buclă care se repetă la nesfârșit?
   a. O buclă pixel.
   b. O buclă spartă.
   c. O buclă infinită.
   d. O buclă ‘iffy’.

3. Pentru a număra anumite elemente, folosim în mod normal:
   a. o buclă loop
   b. numere aleatorii
   c. o buclă while
   d. o buclă for

4. Pentru a repeta anumite acțiuni de mai multe ori, fără a număra repetările, folosim în mod normal:
   a. o buclă loop
   b. numere aleatorii
   c. o buclă while
   d. o buclă for

CAPITOLUL 13 –
BIBLIOTECI

La finalul acestui capitol, veți putea să:

- Înțelegeți termenul de eveniment și scopul unui eveniment în cadrul unui program.
- Utilizați evenimentele apărute la acționarea butoanelor mouse-ului, la apăsarea diverselor taste de pe tastatură, la apăsarea unui buton de comandă, temporizator
- Utilizați bibliotecile generice disponibile, precum: math, random, time

13.1 UTILIZAREA BIBLIOTECILOR

Concepte

Funcțiile și procedurile oferă o modalitate de reutilizare a codului. Alte modalități de reutilizare a codului pe o scară mai largă sunt:

Biblioteci

O bibliotecă reprezintă o colecție de module ce conțin proceduri și funcții deja scrise, ce pot fi reutilizate în cadrul programelor. Astfel se economisește mult timp și multă muncă de scriere a procedurilor și funcțiilor de la început.

- Codul sursă al funcțiilor din modulele bibliotecii nu este vizibil în program
- Un modul dintr-o bibliotecă este adăugat prin scrierea numelui acesteia în comanda 'import'.

Modulul ECDL Computing se bazează pe utilizarea următoarelor module din biblioteca standard:

Random, Math și Time
Aceste module sunt incluse în Python la instalarea mediului IDLE și care furnizează funcții pentru lucrul cu numere.
pygame
O bibliotecă opțională ce conține module cu funcții și proceduri pentru lucrul cu grafice și animații.

Cod Boilerplate
Reprezintă codul utilizat în cadrul unui proiect fără modificări sau cu modificări minore. De exemplu, acest cod poate furniza funcții pentru afișarea imaginilor pe mai multe pagini ale unui document.

- Codul Boilerplate este un cod sursă vizibil în program.
- Codul Boilerplate este utilizat prin includerea tuturor codurilor sale sursă în programul la care se lucrează.

La utilizarea unui modul al unei biblioteci sau a unui cod Boilerplate, nu este necesar să cunoașteți detaliile legate de modul de funcționare al algoritmilor din acestea.

Acest material include cod boilerplate pentru inițializarea și utilizarea modulelor din biblioteci.

13.2 BIBLIOTECI STANDARD

Concepte

Instrucțiuni de import

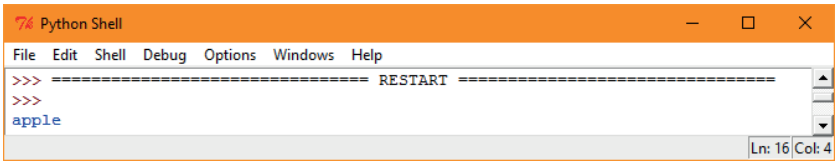
Python utilizează instrucțiunea de import pentru a putea utiliza codul din anumite module ale bibliotecilor (standard și Pygame). Există 2 versiuni ale instrucțiunii de import:

Prima versiune utilizează cuvântul cheie **import** urmat de numele modulului.

Când utilizați această versiune de import, funcțiile au întâi numele modulului, apoi un punct și apoi numele funcției. Un exemplu este prezentat mai jos.

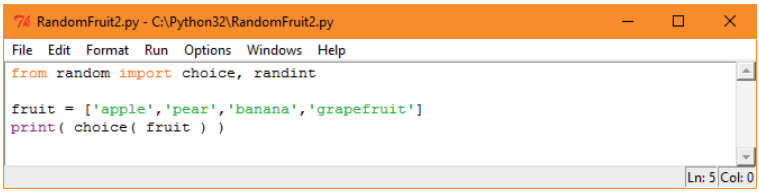
Exemplu de instrucțiune import fără denumirea funcțiilor

Rularea acestui exemplu va returna numele unui fruct aleator, așa cum este prezentat în exemplul de mai jos:



O versiune alternativă de instrucțiune de import numește fiecare funcție care trebuie importată. Ea utilizează cuvântul cheie **from**, apoi numele modulului, apoi cuvântul cheie **import**, urmat de numele funcțiilor care trebuie importate, separate prin virgulă.

Când utilizați această versiune de import, funcțiile nu trebuie să aibă înaintea lor numele modulului bibliotecii. Un exemplu este prezentat mai jos.



Exemplu de instrucțiune import cu denumirea specifică a funcțiilor

3 Module ale bibliotecii Standard

Există 3 biblioteci standard predefinite în Python:

MODUL	SCOP
time	Funcții legate de timp, cum ar fi pentru afișarea datei, orei curente sau zilei din săptămână.
random	O bibliotecă pentru generarea numerelor aleatorii.
math	O bibliotecă pentru funcții matematice.

Modulul time

Există 2 funcții extrem de utile în modulul time. În general, cele 2 funcții sunt utilizate împreună:

FUNCȚIE	SCOP
strftime	Această funcție preia un obiect al bibliotecii time și îl convertește într-un șir de caractere. Această operație poartă numele de formatare. Există mai multe opțiuni de formatare disponibile.
gmtime	O funcție care returnează un obiect de tip time pentru ora curentă.

Funcția strftime utilizează 'specificatori de format' pentru a descrie exact cum ar trebui să arate șirul rezultat. Spre exemplu, literele '%Y' sunt înlocuite cu anul și ca urmare șirul 'În anul %Y' ar putea fi modificat în șirul 'În anul 2001' prin funcția strftime.

Regăsiți mai jos un tabel al formatelor disponibile pentru funcția strftime:

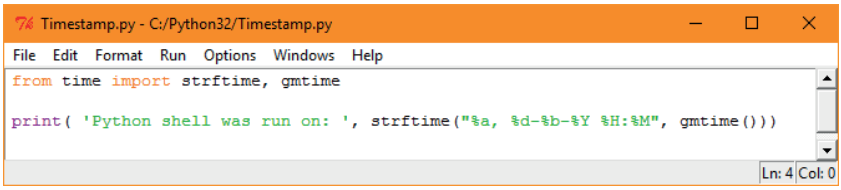
FORMAT	SEMNIFICAȚIE	EXEMPLU
%a	Zi a săptămânii (scurt)	Mon
%A	Zi a săptămânii (lung)	Monday
%b	Numele lunii (scurt)	Sep
%B	Numele lunii (lung)	September
%c	Data și ora	15/09/76 17:22:56
%d	Data	15
%H	Ora (format 24 ore)	17
%I	Ora (format 12 ore)	5
%j	Zi din an	350
%m	Numărul lunii	9
%M	Minute	22
%p	AM sau PM	PM
%S	Secunde	56
%u, %w or %W	Număr săptămână	15
%x	Dată	15/09/76
%X	Oră	17:22:56

FORMAT	SEMNIFICAȚIE	EXEMPLU
%y	An (2 cifre)	76
%Y	An (4 cifre)	1976
%Z	Zonă Timp	GMT Standard Time

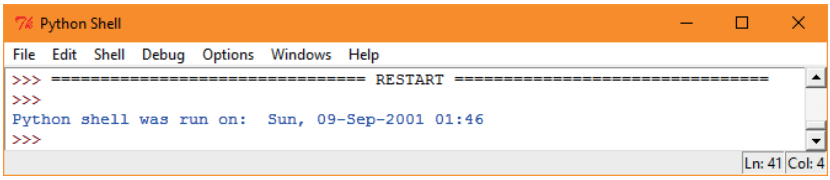
Exemplu: Marcaj de timp

În acest exemplu, funcția strftime este utilizată pentru afișarea unui marcaj de timp. Acest exemplu utilizează multe dintre opțiunile de formatare din tabelul prezentat mai sus.

1. Creați un nou program numit Timestamp.py.
2. Scrieți codul de mai jos.



3. Rulați programul pentru a obține următorul rezultat. Data și ora afișate vor fi cele curente, nu cele afișate în imaginea de mai jos.



Modulul random

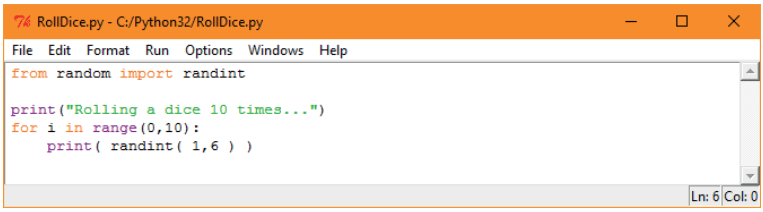
Cele 2 funcții utile în modulul random sunt choice și randint.

FUNCȚIE	SCOP
choice	O funcție care alege un element aleator dintr-o listă
randint	O funcție care alege un număr întreg aleator între 2 numere întregi. randint(1,6), spre exemplu, alege un număr aleator între 1 și 6, exact ca și cum ai arunca un zar.

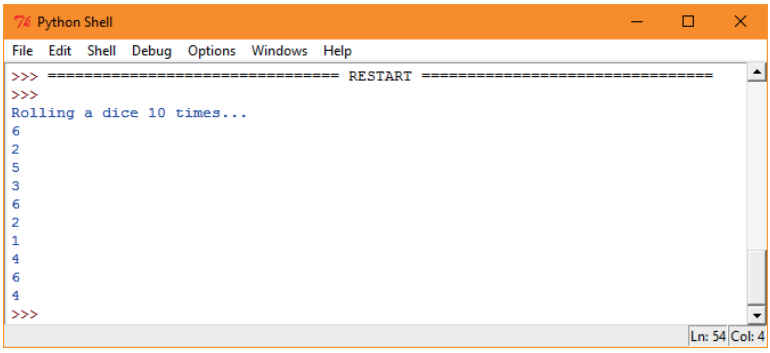
Exemplu: Aruncarea zarurilor

În acest exemplu, funcția randint este utilizată pentru a arunca un zar de 10 ori.

1. Creați un nou program numit RollDice.py.
2. Scrieți codul de mai jos.



3. Rulați programul.



Modulul math

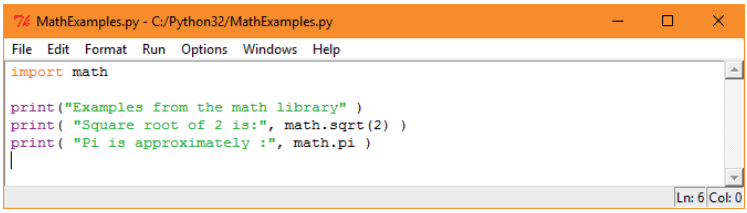
Unele funcții din modulul math sunt:

FUNCȚIE	SCOP
log10	Logaritm în bază 10. log10(100000) este 5. Numără numărul de zerouri.
pow	Ridică la putere. pow(10,3) înmulțește 10 cu 10 de 3 ori, așa că rezultatul este 1000.
sqrt	Rădăcină pătrată. sqrt(16) este 4. sqrt(25) este 5.
sin	Sinusul unui unghi măsurat în radiani.
cos	Cosinusul unui unghi măsurat în radiani.
tan	Tangenta unui unghi măsurat în radiani.
factorial	Număr factorial. factorial(5) este 5 * 4 * 3 * 2 * 1 adică120

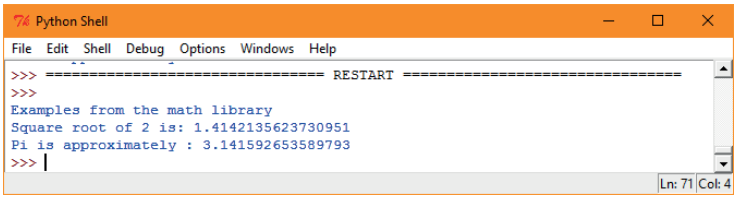
În plus, modulul math conține valoarea pentru constanta matematică pi.

Exemplu: utilizarea modulului Math

1. Creați un nou program numit MathExample.py.
2. Scrieți codul de mai jos.



3. Rulați programul.



13.3 EVENIMENTE

Concepte

Fluxul unui program poate depinde de desfășurarea anumitor acțiuni. Utilizatorii pot declanșa aceste acțiuni prin executarea unui click de mouse pe un grafic de pe ecran, prin apăsarea unei taste de pe tastatură sau prin accesarea unui link într-un browser web. În Python, aceste acțiuni poartă numele de evenimente.

Evenimentele sunt utilizate exclusiv în jocuri. Evenimentele care determină mișcarea unui personaj într-un joc sunt declanșate de o acțiune a utilizatorului, cum ar fi: utilizarea săgeților de direcție pentru a muta personajul într-o anumită direcție, utilizarea unui buton al mouse-ului pentru a face personajul să sară, etc.

Anumite evenimente pot fi specifice părții de hardware. În cadrul unui sistem de alarmă ce utilizează un computer pentru a monitoriza intrușii, o persoană care se mișcă în apropierea unui senzor poate cauza sau declanșa un eveniment.

În unele computere, atunci când bateria atinge un nivel minim de încărcare se poate declanșa un eveniment și anume afișarea unui mesaj de avertizare legat de înlocuirea bateriei.

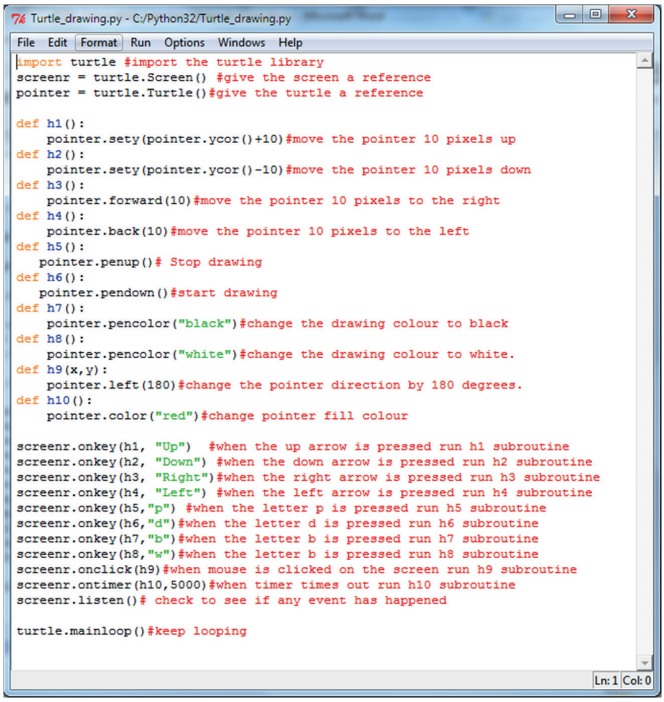
Evenimente

Un handler de eveniment (event handler) reprezintă o porțiune de cod proiectată să execute o acțiune atunci când se declanșează un anumit eveniment. De exemplu, săgețile de direcție de pe tastatură pot controla mișcarea unui jucător în cadrul unui joc video. În funcție de ce eveniment are loc (sus, jos, stânga, dreapta), programul va executa codul 'handler-ului de eveniment' adecvat, care va realiza mutarea corectă a jucătorului.

Unele limbaje de programare au un sistem de 'înregistrare' a funcțiilor care sunt apelate atunci când se întâmplă diverse evenimente. Putem recunoaște aceste funcții după numele lor care începe cu cuvântul 'On' urmat de numele evenimentului. Spre exemplu, 'OnClick' este apelată atunci când utilizatorul generează un eveniment executând click oriunde pe ecran.

Exemplu: Explorarea handler-elor de evenimente

Python are în biblioteca standard un modul numită turtle. Este un simplu pachet inclus în instalarea aplicației Python. Pentru a utiliza funcțiie și procedurile pachetului (modulului) trebuie să utilizați comanda de import.



Cum funcționează programul:

Comentariile din program explică ce face fiecare din cele 10 subrutine. Programul conține o buclă continuă, verificând la fiecare trecere dacă un eveniment a fost declanșat. Atunci când un eveniment a fost declanșat, programul apelează o subrutină atribuită respectivului eveniment. Când termină, revine la corpul principal al programului.

- Primele 4 handlere de evenimente onkey sunt utilizate pentru a determina dacă este apăsată vreo tastă de direcție de pe tastatură. Subrutinele h1-h4 sunt invocate imediat ce este detectată o apăsare a unei taste. Acestea sunt evenimente generate de tastatură.
- Următoarele 4 handlere de evenimente onkey sunt utilizate pentru a detecta dacă sunt apăsate tastele p,d,b sau w de pe tastatură, apelând subrutinele h5-h8. Acestea sunt evenimente generate de tastatură.
- Următorul handler de evenimente este utilizat pentru a detecta un click de mouse și apoi a rula subrutina h9. Acesta este un eveniment generat de mouse.
- Ultimul handler de evenimente este utilizat pentru a declanșa un eveniment atunci când cronometrul se oprește. După scurgerea timpului, este rulată subrutina h10.

Notă: comanda "screenr.listen" este utilizată pentru verificarea altor evenimente.

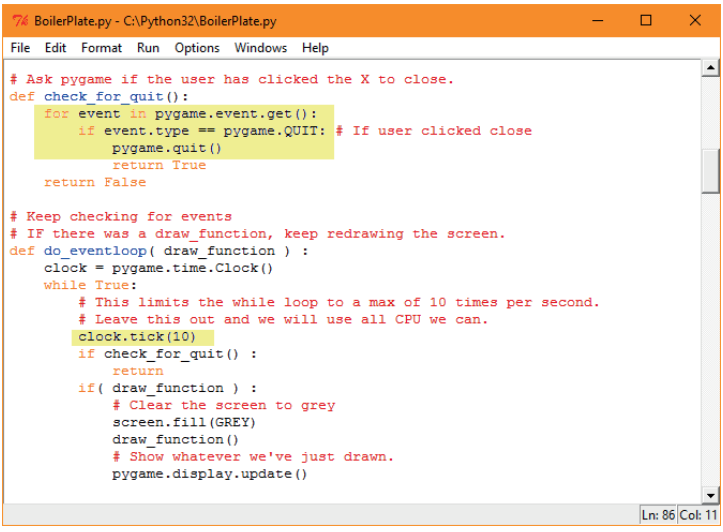
- Introduceți codul din imaginea de mai sus, fără comentarii.
- Salvați programul cu numele Turtle_drawing.py.
- Rulați programul.
- Remarcați schimbarea cursorului din negru în roșu după 5 secunde. Acesta reprezintă declanșarea evenimentului din program de către un temporizator.

- Remarcați că atunci când apăsați tastele de direcție de pe tastatură, cursorul se deplasează și trasează linii. Acestea sunt evenimente declanșate de apăsarea fiecărei taste de direcție de pe tastatură.
- Apăsarea tastei p de pe tastatură dezactivează stiloul, acesta fiind un alt eveniment generat de tastatură.
- Apăsarea tastei d de pe tastatură activează stiloul, acesta fiind un alt eveniment generat de tastatură.
- Apăsarea tastei b sau w de pe tastatură modifică culoarea cernelii stiloului, acesta fiind un alt eveniment generat de tastatură.
- În final, executați click pe mouse oriunde pe ecran. Cursorul mouse-ului se va roti cu 180 de grade. Tocmai ați declanșat un eveniment generat de mouse.

13.4 BIBLIOTECA PYGAME

Concepte

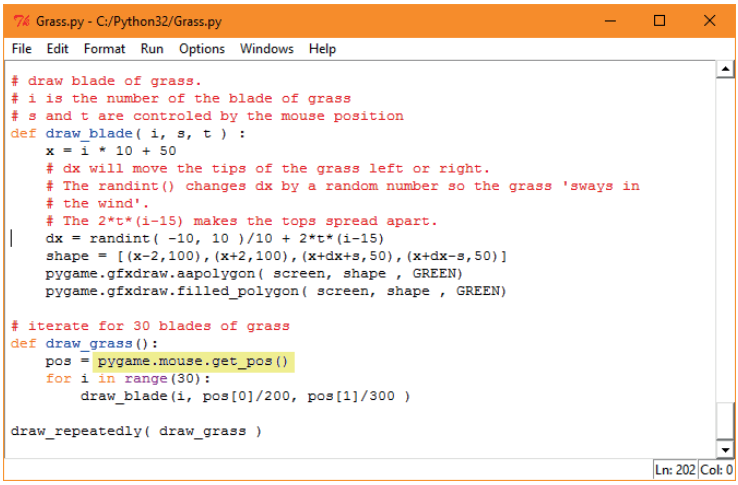
În Python, folosind Pygame, codul pentru gestionarea evenimentelor este scris astfel încât să se solicite în mod repetat informații de verificare a evenimentelor care au avut loc. Imaginea de mai jos se referă la exemplul legat de firele de iarbă descris mai târziu în acest capitol. Pentru moment, să aruncăm o privire pe cele 2 evenimente evidențiate mai jos.



Gestionarea evenimentelor evidențiate în fișierul BoilerPlate.py

În funcția check_for_quit(), o buclă for verifică în mod repetat dacă evenimentul Quit a avut loc în Pygame. Evenimentul QUIT este declanșat atunci când utilizatorul apasă pe butonul X aflat în colțul din dreapta sus al ferestrei Pygame. Dacă acel eveniment este detectat, funcția check_for_quit() returnează valoarea True. Dacă evenimentul nu este detectat, funcția check_for_quit() returnează valoarea False.

A doua porțiune evidențiată de cod, apelarea funcției clock.tick(), este relaționată de asemenea cu gestionarea evenimentelor; ea stabilește frecvența de rulare a buclei while, care la rândul ei stabilește frecvența de actualizare a ecranului.



Gestionarea evenimentelor evidențiate în fișierul Grass.py

Un alt exemplu de eveniment evidențiat mai sus este procedura draw_grass(). Este apelată mouse.get_pos() pentru a determina poziția mouse-ului, care la rândul său este folosită pentru a determina poziția firelor de iarbă.

13.5 COD BOILERPLATE

Concepte

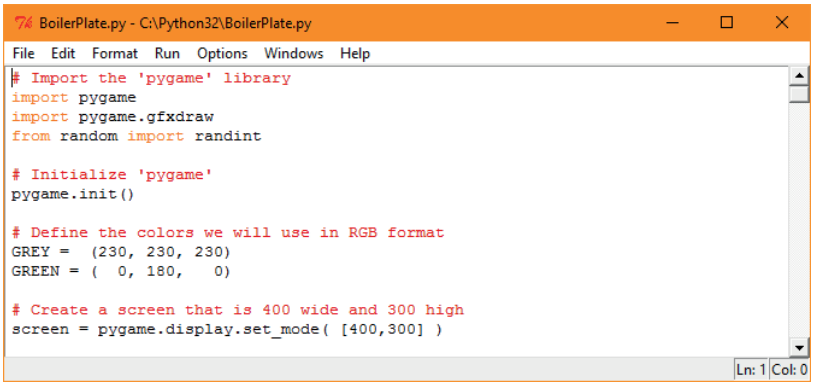
Pentru utilizarea bibliotecii Pygame, un program trebuie să adauge această bibliotecă, să o inițializeze, să definească culorile și să execute o serie de alte acțiuni. Codul Boilerplate este oferit în acest material tocmai pentru a executa aceste acțiuni.

Nu este necesar să cunoaștem cum funcționează în detaliu codul boilerplate. Trebuie să știm doar că:

- Tuplurile sunt utilizate pentru a defini o serie de culori folosite la pictarea ecranului.
- Codul boilerplate importă din modulul random al bibliotecii standard pentru a obține funcția randint.
- Are un eveniment buclă.
- Utilizează randint pentru a genera un număr întreg aleator.

Exemplu: Explorarea fișierului BoilerPlate.py

1. Localizați și deschideți fișierul BoilerPlate.py furnizat împreună cu acest material. (Fișierele de lucru se pot descărca de pe site-ul www.ecdl.ro)



2. Priviți liniile aflate după comentariul # Import the 'pygame' library. Toate cele 3 linii conținând cuvântul 'import' se referă la utilizarea bibliotecilor:
   - 'import pygame' permite accesul la biblioteca pygame astfel încât aceasta să devină disponibilă spre a fi utilizată.
   - 'import pygame.gfxdraw' permite accesul la o extensie opțională a pygame care desenează diverse forme mai netede.
   - modulul 'random' furnizează funcții pentru lucrul cu numere aleatorii. Comanda de import permite accesarea funcției randint pentru a genera numere întregi aleatorii.

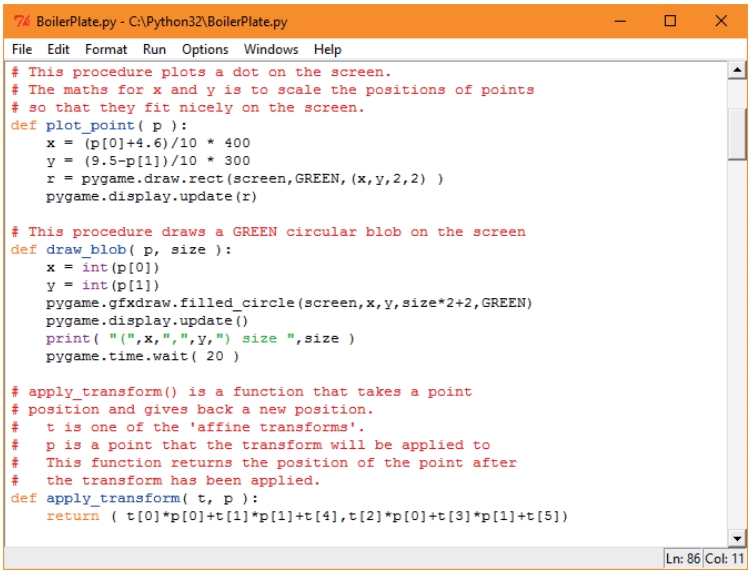
   Restul de cod:

   - inițializează biblioteca Pygame. Aceasta conține multe variabile care trebuie să aibă valori inițiale.
   - definește și inițializează 2 culori, gri și verde. Parantezele rotunde indică faptul că aceste culori sunt definite ca tupluri cu 3 valori întregi.
   - crează o variabilă numită screen ce reprezintă o fereastră de 400 de pixeli lățime și 300 de pixeli înălțime.
3. Derulați – codul boilerplate definește unele proceduri pentru reprezentarea grafică a punctelor și pensulelor. Citiți comentariile care descriu fiecare subrutină.

Subrutinele sunt explicate mai jos mai detaliat decât în comentarii. Această secțiune de cod furnizează funcții și proceduri utilitare.

- plot_point va reprezenta grafic un punct pe ecran. Utilizează numerele 400 și 300, reprezentând lățimea și înălțimea ecranului, pentru a scala punctul în poziția corectă pe ecran.
- draw_blob va desena pe ecran o pensulă verde circulară. De asemenea, va afișa coordonatele pensulei.
- apply_transform va aplica punctului o 'transformare affine (polinomială de ordinul I)'. Aceasta este utilizată pentru distorsionarea formelor. Poate mări/micșora, roti, reflecta o formă, o poate muta la stânga sau la dreapta, sus sau jos. Pentru acest modul, este suficient să știm faptul că transformarea affine preia anumite coordonate și returnează altele ca și rezultat.

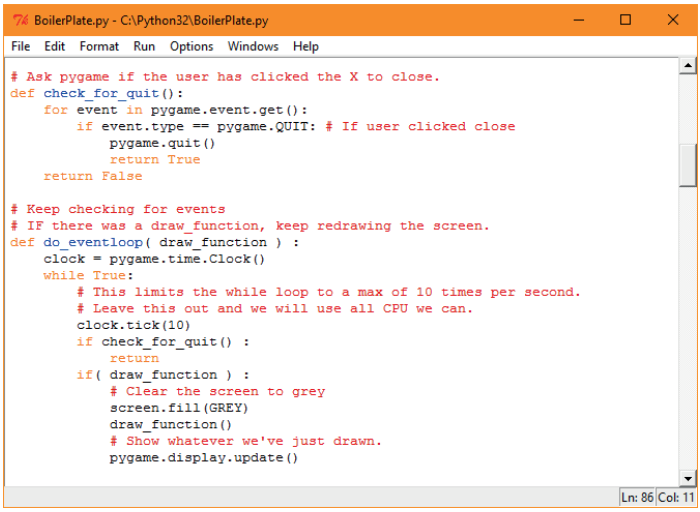
Imaginea de mai jos este prezentată ca exemplu al comentariilor și structurii codului din secțiunea anterioară.



4. Derulați – următoarea partea a codului boilerplate se referă la evenimente.
   - Citiți comentariile din codul de procesare a evenimentelor.
   - Remarcați faptul că check_for_quit() este o funcție. Ea returnează o valoare booleană.
   - Remarcați faptul că do_eventloop() este o procedură. Nu conține cuvântul cheie return, dar comanda return nu returnează o valoare.

Codul de procesare a evenimentelor este:

- check_for_quit() verifică un eveniment. El interoghează Pygame dacă utilizatorul a apăsat pe butonul 'X' din partea din dreapta sus a ferestrei. Dacă da, este momentul ca acea fereastră să fie închisă.
- do_eventloop() are o buclă infinită, 'while True', care verifică în mod continuu comanda de închidere a ferestrei și redesenează ecranul grafic. Desenarea repetată este relevantă pentru animație.
- do_eventloop() utilizează o tehnică neobișnuită. Parametrul transferat în do_eventloop() poate fi o funcție. Dacă este într-adevăr o funcție, atunci do_eventloop va apela această funcție de fiecare dată când va desena ecranul grafic.



5. Derulați

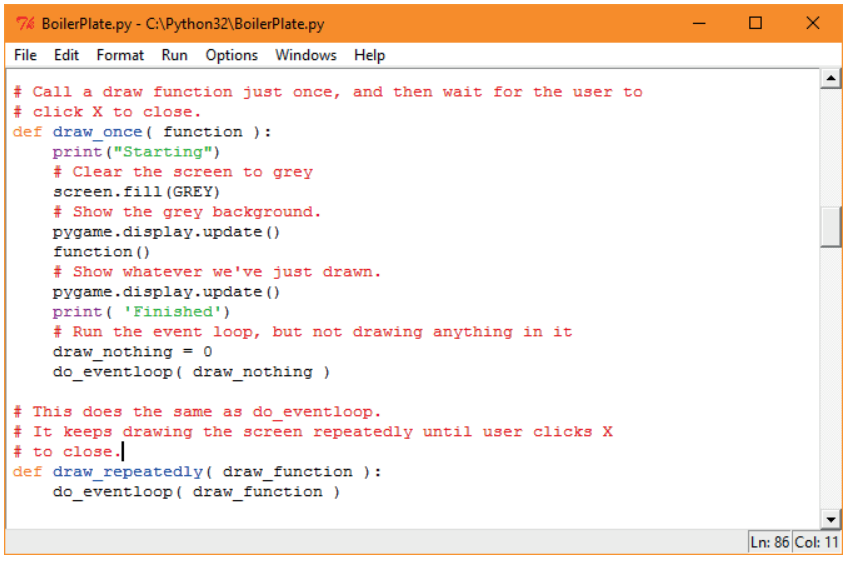
Restul de cod oferă 2 modalități de desenare a ecranului.

Citiți comentariile pentru proceduri.

Aceste 2 proceduri oferă 2 opțiuni de utilizare a unei bucle eveniment.

- draw_once() va rula codul de desenare o singură dată, apoi va aștepta ca utilizatorul să părăsească ecranul grafic.
- draw_repeatedly() va rula codul de desenare la nesfârșit, verificând de fiecare dată dacă utilizatorul a părăsit sau nu ecranul grafic. Această procedură este potrivită pentru animații.

Funcția draw_once() conține o secvență de acțiuni. Aceasta include afișarea mesajelor 'Starting' și 'Finished' și așteptarea comenzii de închidere în bucla eveniment.



Exemplu: Realizarea unei copii a fișierului BoilerPlate.py

Pentru a utiliza Pygame, realizați o copie a codului boilerplate și apoi adăugați codul dvs.

1. Deschideți fișierul BoilerPlate.py.
2. Utilizați pașii de la exemplul 'Crearea și salvarea unui program'. Utilizați pașii de la Save As încolo pentru a salva fișierul BoilerPlate.py cu un nou nume ales de dvs.
3. Verificați că ați realizat copia dorită deschizând documentul salvat la pasul anterior.

13.6 DESENAREA UTILIZÂND BIBLIOTECILE

Concepte

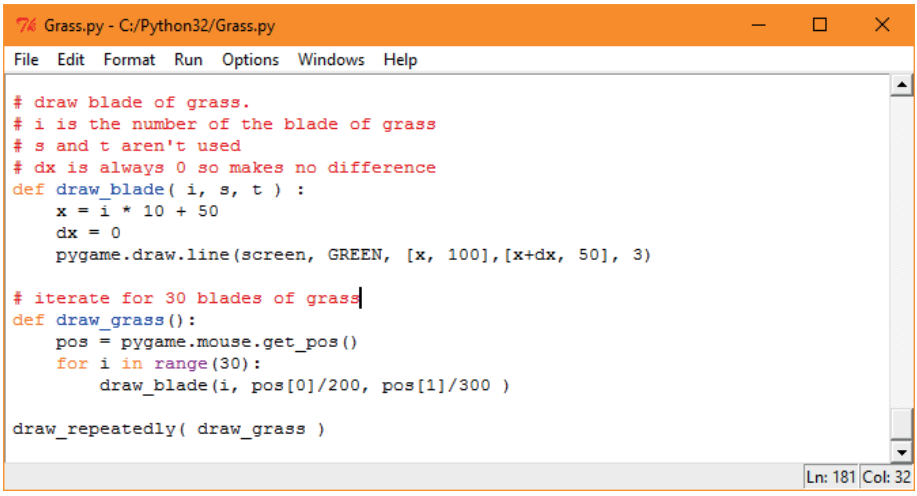
Pentru jocurile 3D profesionale, graficienii crează artă personalizată pentru majoritatea caracteristicilor lumilor și monștrilor din joc. Cu toate acestea, foarte multe detalii sunt generate pe baza codului de programare. Spre exemplu, graficienii nu desenează fiecare fir de iarbă și nu crează animație pentru fiecare fir de iarbă care se mișcă în bătaia vântului. Ei crează în loc una sau mai multe tipuri de fire de iarbă și apoi utilizează proceduri și bucle pentru a desena și anima mai multe fire de iarbă.

Pygame are o bibliotecă grafică 2D. Codul Pygame poate utiliza o buclă și numere aleatoare pentru a demonstra faptul că un computer poate executa instrucțiuni repetitive. Codul poate desena și anima un singur fir de iarbă, după care codul poate fi introdus într-o buclă pentru a desena și anima multe fire de iarbă.

În următorul exemplu Pygame 'firele de iarbă' sunt doar niște linii verzi.

Exemplu: Fire de iarbă

1. Realizați o copie a codului boilerplate.py
2. Denumiți fișierul copie Grass.py
3. Adăugați la final următorul cod:



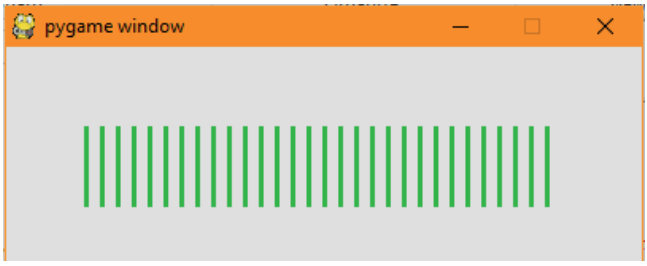
4. Reverificați următoarele puncte cu privire la acest cod. Verificați dacă informațiile de mai jos sunt relaționate cu codul actual.

draw_grass() conține o buclă și apelează draw_blade() de 30 de ori.
draw_blade() la rândul său apelează draw_line() în biblioteca pygame și desenează o linie verde de grosime 3.

Modificând verde să fie tuplul (0,0,255) ar fi generat iarbă albastră.

- draw_blade() conține 3 parametri, i, s și t, dar în această etapă este utilizat doar un singur parametru, i.
- i reprezintă numărul de fire de iarbă. i stabilește poziția firelor de iarbă, x.
- 'x = i * 10 + 50' derivă x din i. Formula preia numărul firelor de iarbă, ce ar putea fi 0, 1, 2, 3... și îl modifică într-o coordonată x, 50, 60, 70, 80... *10 depărtează firele de iarbă la 10 pixeli distanță și +50 pornește secvența la x=50. Modificați puțin aceste valori și rulați programul pentru a vedea diverse spațieri și poziții de start.
- draw_grass() invocă funcția mouse.get_pos().
- mouse.get_pos() obține poziția mouse-ului sub forma unui tuplu. Acesta conține poziția x și y a mouse-ului. Codul împarte x la 200 și y la 300 pentru a obține valorile parametrilor s și t pentru draw_blade.

5. Rulați programul pentru a vedea firele de iarbă:



În prezent, imaginea cu firele de iarbă conține linii drepte. Pentru a face firele de iarbă mai ascuțite, înlocuiți codul draw_line().

6. Efectuați următoarele modificări codului:

Grass.py - C:/Python32/Grass.py

File Edit Format Run Options Windows Help

```
# draw blade of grass.
# i is the number of the blade of grass
# s and t aren't used
# dx is always 0 so makes no difference
# Now the grass is a bit more pointy.
def draw_blade( i, s, t ) :
    x = i * 10 + 50
    dx = 0
    shape = [[x-2,100],[x+2,100],[x+dx+s,50],[x+dx-s,50]]
    pygame.gfxdraw.aapolygon( screen, shape , GREEN)
    pygame.gfxdraw.filled_polygon( screen, shape , GREEN)

# iterate for 30 blades of grass
def draw_grass():
    pos = pygame.mouse.get_pos()
    for i in range(30):
        draw_blade(i, pos[0]/200, pos[1]/300 )

draw_repeatedly( draw_grass )
```

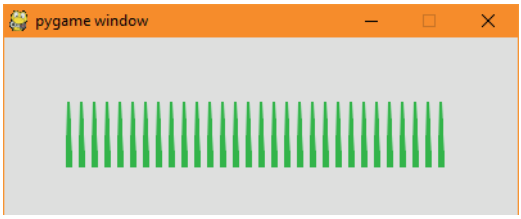
Ln: 180 Col: 0

7. Relaționați descrierea de mai jos cu modificările din cod.

Cea mai mare modificare este crearea unei noi variabile shape.

- Noua variabilă listă 'shape' stochează coordonatele celor 4 colțuri ale firelor de iarbă.
- S-a renunțat la draw.line. Cele 2 comenzi gfxdraw desenează conturul exterior al formei și apoi interiorul. Modificați culoarea liniei de contur în verde.

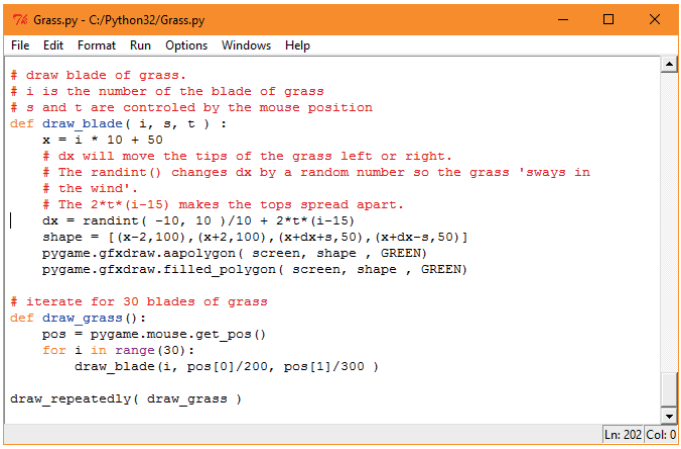
8. Rulați codul pentru a vedea firele mai ascuțite de iarbă.



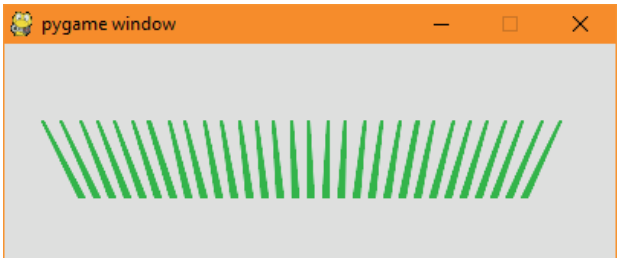
9. Următorul pas realizează 2 modificări suplimentare în program:
   - Face iarba mai distanțată
   - Face iarba să se vălurească

Există o singură modificare majoră a codului. Ea afectează variabila dx. dx deplasează vârful ierbii spre stânga sau dreapta. 0 înseamnă că nu există nicio deplasare spre stânga sau dreapta. Un dx negativ mută vârful spre stânga. Un dx pozitiv mută vârful spre dreapta. Modificarea din cod modifică instrucțiunea dx = 0 pentru a avea o calculație pentru dx.

10. Modificați codul după cum urmează:



11. Relaționați descrierea de mai jos cu modificările din cod.
    - dx deplasează vârfurile ierbii spre stânga sau dreapta.
    - randint(-10, 10) este o funcție din biblioteca random care generează un număr întreg aleator între -10 și +10. Apoi, acesta este împărțit la 10 pentru a obține un număr cu virgulă mobilă între -1 și +1. Așadar, randint() deplasează vârfurile ierbii cu 1 pixel aleator spre stânga sau dreapta.
    - 2*t*(i-15) nu este aleator. Face ca vârfurile ierbii să se împăștie. Dacă această porțiune a expresiei este omisă din cod, firele doar se deplasează aleator, însă nu se împrăștie.
12. Rulați programul pentru a obține următoarea imagine:



Imaginea afișată nu arată că iarba se mișcă acum, însă este vizibil faptul că iarba este acum mai distanțată.

13.7 EXERCIȚII RECAPITULATIVE

1. Un eveniment este:
   a. Atunci când un program rulează și se deplasează din codul scris de programator în codul unei biblioteci.
   b. Un lucru care se întâmplă și la care programul ar trebui să reacționeze.
   c. Atunci când un program conține prea multe date și trebuie să renunțe la o anumită parte din aceste date.
   d. Un tip de date special în Python utilizat pentru a urmări date importante, cum ar fi aplicațiile de tip calendar.

2. În ce modul (pachet) al bibliotecii standard se găsește 'randint'?
   a. math.
   b. randint.
   c. monster.
   d. random.

3. Șirul numit greeting are valoarea ‘Good’. Ce comandă îi stabilește valoarea la ‘Good Morning’?
   a. greeting = ‘Good’ + greeting
   b. greeting = greeting + “ Morning”
   c. greeting = ‘Good’ + ‘Morning’
   d. greeting = [‘Good’, ‘Morning’]

4. Tuplul numit colour are valoarea (0,0,255). Ce comandă îi stabilește valoarea (255,0,255)?
   a. colour[0] = 255
   b. colour[1] = 255
   c. colour = (255 , 0+0 , 255)
   d. colour = 255,0,255

5. Care dintre următoarele nu generează evenimente în Python cu Pygame?
   a. Executarea unui click pe mouse.
   b. Un temporizator.
   c. Introducerea unor date de la tastatură.
   d. Descărcarea bateriei.

CAPITOLUL 14 –
RECURSIVITATE

La finalul acestui capitol, veți putea să:

- Înțelegeți termenul de recursivitate

Această lecție consolidează, de asemenea, conceptele din lecțiile anterioare, cum ar fi:

- Buclă
- Procedură
- Funcție
- Variabilă
- Parametru

14.1 RECURSIVITATE

Concepte

Exemplul cu iarba mișcătoare din capitolul 13 și exemplul cu săgeata din capitolul 12 implică subrutine care apelează alte subrutine. Aceasta este una dintre metodele de descompunere a unei probleme în programare. Țineți minte că descompunerea se referă la divizarea unei probleme complexe în probleme mai mici. Apoi, subrutinele sunt apelate și executate până când problema globală este rezolvată.

Este posibil de asemenea ca o subrutină să se apeleze singură o dată sau de mai multe ori. Cu condiția să existe o protecție împotriva unei subrutine care se va apela singură la nesfârșit, subrutinele care se apelează singure pot reprezenta o tehnică puternică pentru descompunerea anumitor probleme.

O subrutină care se apelează singură este cunoscută sub numele de Recursivitate.

Recursivitatea reprezintă procesul prin care o subrutină divizează o problemă complexă în părți mai simple și apoi se apelează singură pentru rezolvarea părților simple. O funcție recursivă se apelează singură în cadrul funcției.

Atunci când o funcție recursivă se invocă pe ea însăși, se spune că **funcția se repetă în mod nelimitat**.

Tehnica recursivității este cea mai utilă atunci când o problemă poate fi descompusă în 2 probleme mai mici, dar care sunt totuși extrem de asemănătoare cu problema originală.

Exemplu: Căutare

Un exemplu de rezolvare a unei probleme utilizând recursivitatea o reprezintă căutarea. O subrutină de căutare ar putea separa posibilitățile de căutare în 2 zone. Subrutina s-ar putea invoca singură pentru a căuta în fiecare zonă, apoi să o împartă și pe aceea și tot așa.

Imaginați-vă verificarea unei case. Algoritmul ar împărți problema în verificarea camerelor și apoi ar împărți verificarea camerelor în verificarea anumitor zone din camere. Eventual zona de verificat ar deveni atât de mică încât poate fi finalizată dintr-un singur pas. Nu este nevoie de o altă subdivizare.

Exemplu: Desenarea unei linii curbe

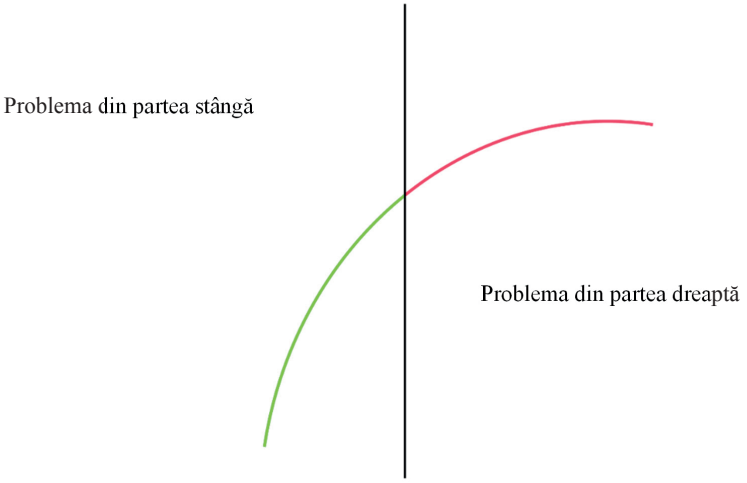
Un alt exemplu al unei probleme ce poate fi rezolvată prin recursivitate ar fi desenarea unei linii curbe:

Un algoritm alege un punct aflat la jumătatea liniei. Acest lucru a împărțit linia curbă în partea stângă și în partea dreaptă.

Împărțirea curbei a descompus problema originală în 2 probleme mai mici. O problemă desenează partea din stânga a liniei curbe, iar cealaltă problemă desenează partea din dreapta.

Aceasta ar putea să nu pară o îmbunătățire a eficienței, dar poate fi, dacă împărțim fiecare jumătate din linia curbă în două probleme mai mici și așa mai departe.

Prin divizarea repetată a problemei, partea liniei curbate care trebuie desenată va deveni eventual suficient de mică încât să încapă într-un singur punct, un pixel pe ecran.



Cum funcționează programul:

- O subrutină recursivă pentru desenarea unei linii curbe se poate apela singură pentru desenarea jumătăților din stânga și dreapta.
- Poate conține un test logic care detectează atunci când o problemă ajunge la dimensiunea unui pixel sau chiar mai mică în dimensiune.
- Atunci când o problemă a fost redusă la o dimensiune atât de mică, subrutina poate desena punctul respectiv în loc să mai facă alte subdivizări.
- Rezultatul combinat al tuturor punctelor formează linia curbă.

14.2 DESEN RECURSIV

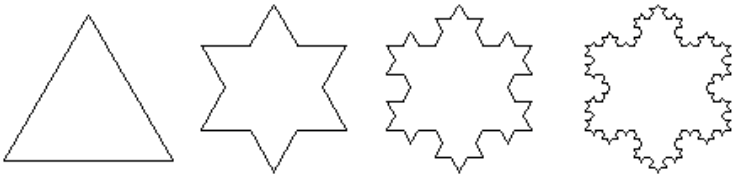
Concepte

Algoritmul de desenare a liniei curbe desenează o dată partea stângă, o dată partea dreaptă. Recursivitatea nu trebuie întotdeauna să împartă o problemă în 2, ea poate împărți problema în mai multe părți. Exemplul următor de desenare a unei frunze de ferigă divizează problema în 4 alte probleme la fiecare etapă.

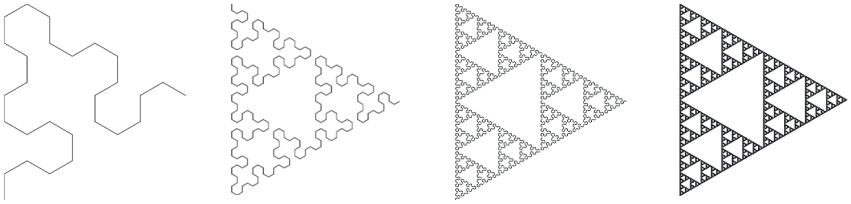
Să analizăm mai întâi un fractal înainte de a continua.

Un **fractal** reprezintă o formă care este similară cu ea însăși, la diferite scale. Dacă mărești un fractal, poți găsi șabloane similare cu șabloanele formei întregi.

Mai jos sunt prezentate câteva exemple ale modului în care un fractal se dezvoltă:



Fractali



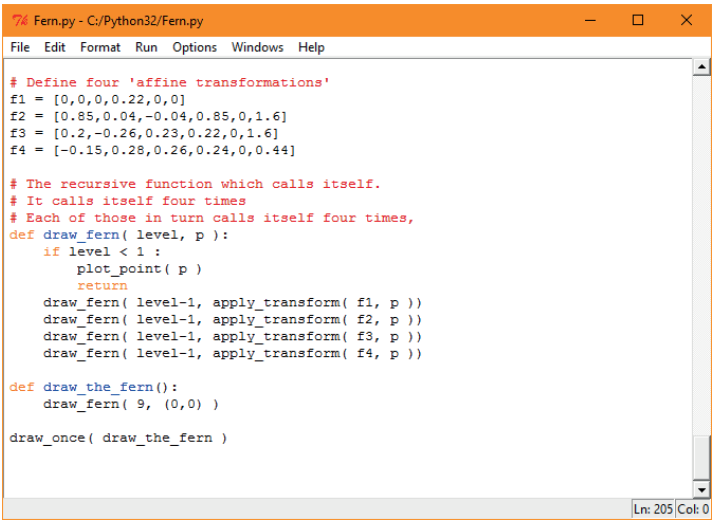
Următorul exemplu utilizează recursivitatea pentru a desena un **fractal** care este o frunză de ferigă**.**

Cum funcționează programul:

- Algoritmul de desenare a fractalului din exemplul de mai jos recurge la desenarea a 4 părți diferite.
- Cele patru opțiuni diferite crează versiuni mai mici, distorsionate ale întregului model în patru locuri diferite. La rândul său, patru versiuni mai mici, distorsionate sunt create în fiecare dintre aceste locuri
- Detaliile celor 4 distorsiuni sunt stocate în variabilele listă numite f1, f2, f3, și f4.
- Codul recursiv, de obicei, conține un parametru care stabilește cât de departe are loc recursivitatea. În acest exemplu acest parametru poartă numele 'level'. De fiecare dată când funcția revine, scade valoarea nivelului. Atunci când nivelul ajunge la 0, algoritmul desenează un punct și oprește subdivizarea mai departe a problemei.

Exemplu: Recursivitatea pentru o frunză de ferigă

1. Realizați o copie de codului boilerplate
2. Denumiți-o Fern.py
3. Adăugați următorul cod:



4. Rulați programul pentru a obține rezultatul de mai jos.

5. Verificați faptul că rularea programului s-a încheiat.

```
>>> ================================ RESTART ================================
>>>
About to draw
Finished
```

Exemplu: Modificarea codului de recursivitate pentru frunza de ferigă

Putem modifica forma ferigii prin schimbarea unor parametri în codul de recursivitate.

```
def draw_the_fern():
    draw_fern( 9, (0,0) )
```

Dacă reducem valoarea 9 din draw_fern(9, (0,0)) la 3, programul va desena mult mai puține puncte.

Dacă creştem valoarea '9' chiar şi un pic, timpul de rulare al programului va creşte simțitor.

Dacă modificați valoarea 9 în 30, execuția programului va dura mai mult de 1 milion de ani.

Sfat: Pentru a opri programul înainte de finalizarea lui, utilizați aceeași tehnică ca la bucla infinită. Mergeți în fereastra Python Shell și apelați meniul 'Shell', opțiunea 'Restart Shell'.

14.3 EXERCIȚII RECAPITULATIVE

1. Un algoritm recursiv este unul care:
    a. Desenează o ferigă.
    b. Desenează o linie curbă.
    c. Utilizează numere aleatorii pentru rezolvarea unei probleme.
    d. Descompune o problemă în părți mai mici și apoi aplică aceeași abordare și acelor părți mai mici.

2. O funcție care se apelează singură:
    a. Este o eroare de secvență incorectă în Python.
    b. Ar putea fi un exemplu de recursivitate.
    c. Va cauza întotdeauna o buclă infinită.
    d. Poartă numele de distorsiune.

3. O buclă infinită este:
    a. Un exemlu de recursivitate.
    b. Un exemplu de distorsiune.
    c. O buclă care rulează la nesfârşit.
    d. O funcție recursivă care desenează cercuri.

CAPITOLUL 15 –
TESTARE ȘI ÎMBUNĂTĂȚIRE

La finalul acestui capitol, veți putea să:

- Înțelegeți tipurile de erori din cadrul unui program precum: erori de sintaxă, erori logice
- Înțelegeți beneficiile testării și depanării unei probleme pentru remedierea erorilor
- Identificați și remediați o eroare de sintaxă într-un program precum: ortografie incorectă, punctuație lipsă
- Identificați și remediați o eroare logică într-un program precum: expresie Booleană incorectă, tip de date incorect
- Verificați dacă programul respectă cerințele din descrierea inițială
- Identificați posibilități de îmbunătățire ale programului astfel încât să satisfacă eventuale nevoi suplimentare
- Descrieți programul complet, comunicând scopul și valoarea

15.1 TIPURI DE ERORI

Concepte

O schemă logică sau un program poate conține erori. Erorile care cauzează funcționarea incorectă a unui algoritm sau program poartă numele de **bug-uri**. Spre exemplu:

- Dacă unul din blocurile de decizie din cadrul unei scheme logice a fost etichetat incorect, cu 'Da și 'Nu' în locuri incorecte, algoritmul va avea un rezultat incorect. Acest tip de eroare poartă numele de bug.
- Dacă în programul cu trucul magic am fi împărțit la 15 în loc de 13, nu am fi obținut rezultatul dorit. Pasul de împărțire la 15 ar fi fost denumit un bug în program.

Bug-urile serioase determină oprirea funcționării programului denumită și **crash**.

Un **crash** reprezintă momentul în care are loc oprirea completă a funcționării programului. Utilizatorii își pot pierde datele ca urmare a unui crash.

În programare, există 3 tipuri de erori frecvente:

O **eroare de sintaxă** are loc atunci când un constructor din limbajul de programare este scris incorect. Este o eroare în cod cauzată de utilizarea incorectă a limbajului, cum ar fi: erori de ortografie, erori de punctuație, utilizarea incorectă a numelor și referințelor. Spre exemplu, (A+3 este o eroare de sintaxă întrucât lipsește paranteza de închidere.

O **eroare logică** are loc atunci când intrucțiunea din program se referă la o acțiune incorectă. Programul pare a fi corect din punct de vedere al sintaxei, (i.e. este scris corect), însă logica este eronată și ca urmare, atunci când este rulat, nu execută ceea ce se presupune că ar trebui. Cu alte cuvinte, un program poate funcționa corect, însă nu respectă cerințele de funcționare.

De exemplu:

```
if test_tube_is_full :
        pour_more_acid_into_it().
```

O **eroare de tip de date incorecte** sau o **eroare de scriere** are loc atunci când se încearcă utilizarea unui operator sau unei funcții pe un tip greșit de obiect. De exemplu, dacă încercăm să înmulțim un șir de caractere cu un număr întreg **3*mere**.

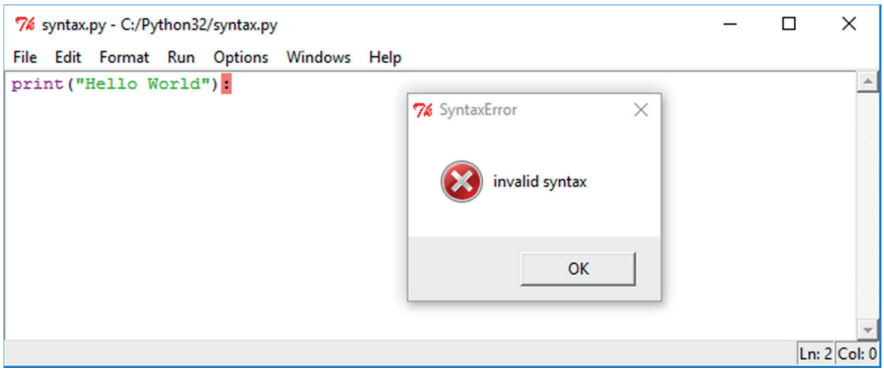
15.2 IDENTIFICAREA ERORILOR

Concepte

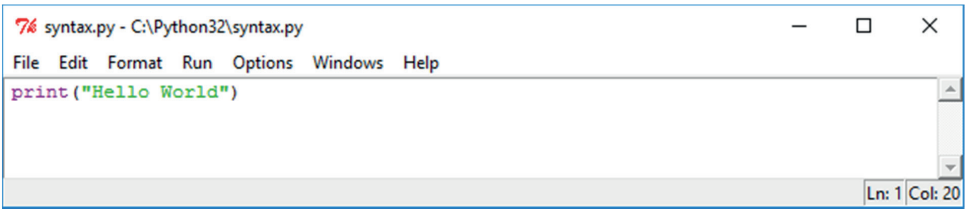
Erori de sintaxă

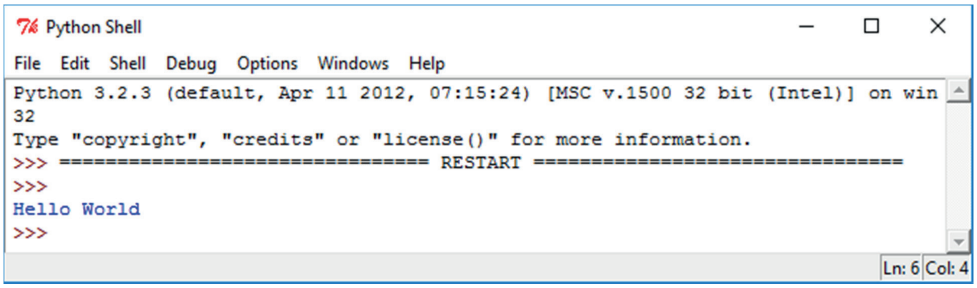
Erorile de sintaxă, cunoscute și sub numele de erori de parcurgere cod, sunt detectate imediat de Python la rularea unui program. Erorile de sintaxă nu sunt considerate bug-uri întrucât Python nu permite rularea programului atunci când identifică o eroare de sintaxă.

Exemple de erori de sintaxă includ sintaxă incorectă, omisiunea unei paranteze sau simbolului :, scrierea greșită a unui cuvând cheie, format incorect al numerelor, etc. Analizorul sintactic, numit și parser, evidențiază primul punct din linie unde este detectată eroarea.



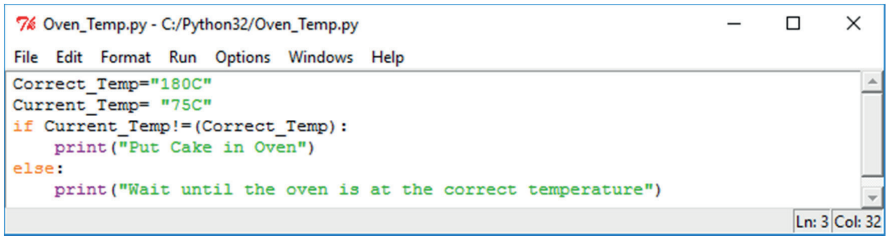
Există semnul : la finalul liniei care este incorect. Corectați această eroare prin ștergerea semnului : și rulați încă o dată programul.

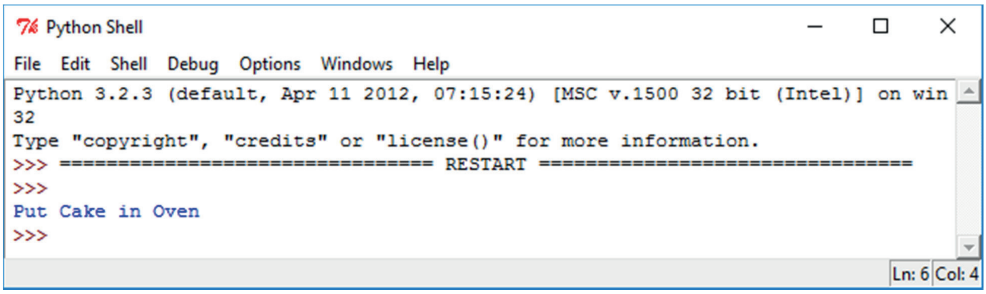




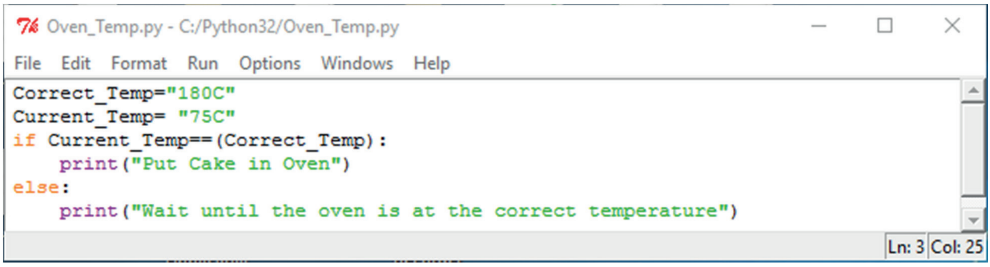
Erori logice

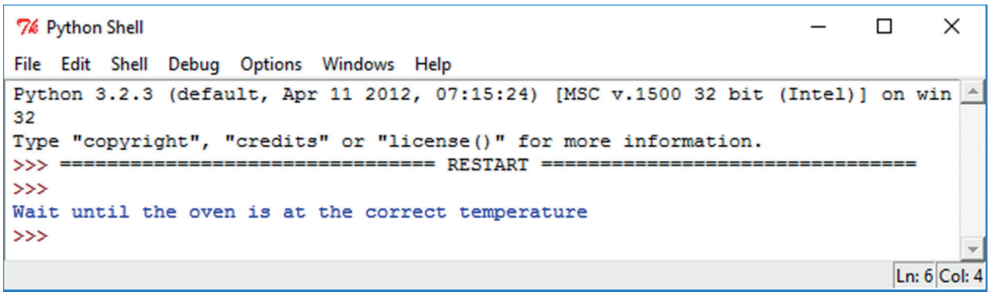
Erorile logice nu sunt întotdeauna ușor de găsit. Analizați logica din acest program pentru a afișa o instrucțiune de introducere a unei prăjituri în cuptor atunci când cuptorul atinge temperatura corectă.



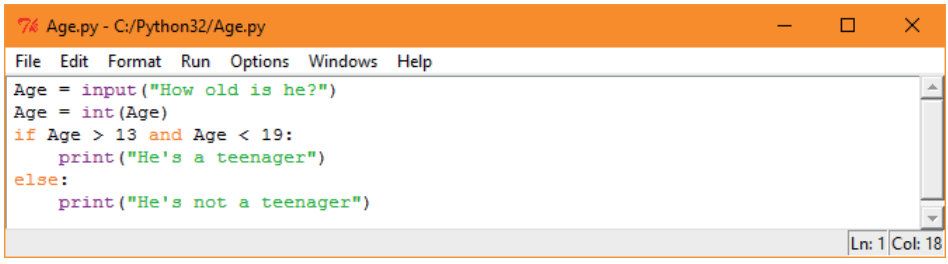


Logica este greșită. Pentru corectarea erorii, trebuie să modificăm simbolul != (Diferit) în == (Egal).





Analizați exemplul de mai jos și verificați dacă puteți corecta eroarea logică:



Există o problemă cu testul logic din acest program.

Testul logic va funcționa corect pentru vârstele 14, 15, 16, 17 și 18 întrucât programul va afișa în mod corect mesajul "He's a teenager" pentru fiecare valoare introdusă.

Însă pentru vârstele 13 și 19 el va afișa mesajul "He's not a teenager"

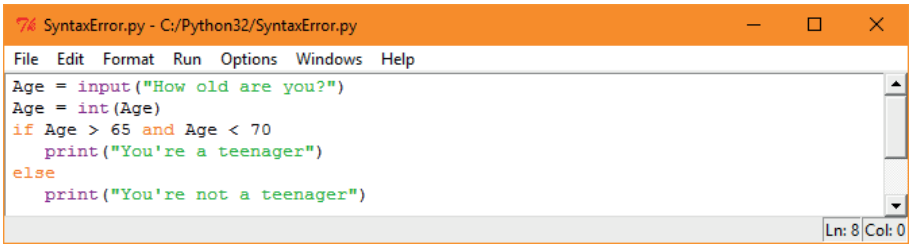
Pentru a remedia eroarea:

1. Deschideți Python.
2. Introduceți codul corect de mai sus pentru a remedia eroarea prin modificarea liniei de cod care începe cu 'if'.
3. Verificați soluția prin rularea programului modificat de dvs.
4. Testați valorile de vârstă 12, 13, 14, 18, 19 și 20 pentru a verifica dacă primiți mesajul corect.

Exemplu: Identificarea și remedierea erorilor logice și de sintaxă

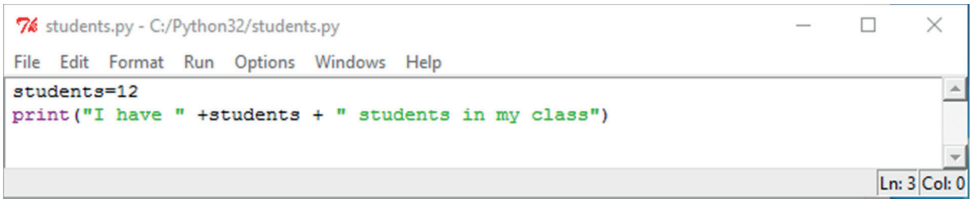
Următorul program conține 2 erori de sintaxă și o eroare logică.

1. Tastați programul de mai jos.

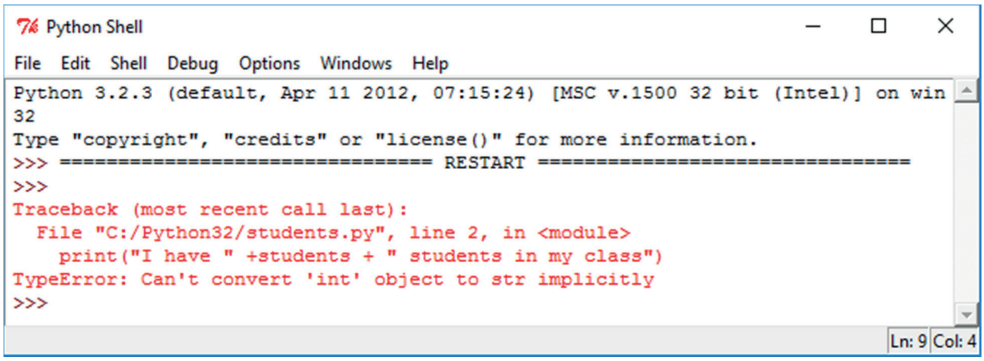


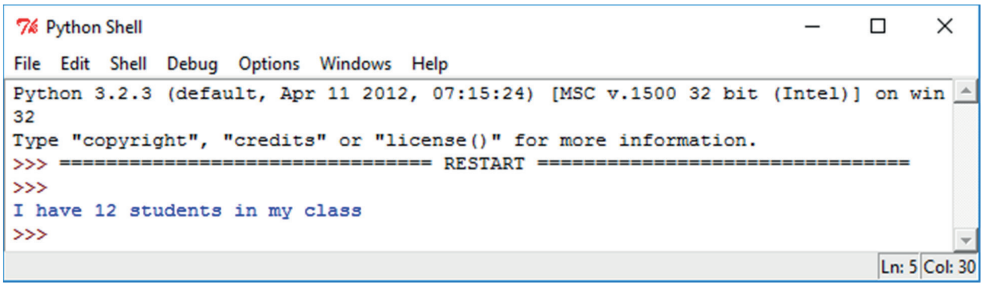
2. Remediați erorile de sintaxă.
3. Remedați de asemenea și eroarea logică
4. Rulați programul.

Exemplu: Identificarea și remedierea unei erori de tip incorect de date



În acest exemplu nu putem asocia la un loc diferite tipuri de date. Nu putem asocia șiruri de caractere cu variabile numerice. Cu toate acestea, putem converti o variabilă numerică într-un șir de caractere. Mai jos, prin utilizarea funcției str() putem converti variabila numerică students într-un șir de caractere.





15.3 TESTAREA ȘI DEPANAREA UNUI PROGRAM

Concepte

Verificarea erorilor

Un programator ar putea remarca sau nu faptul că programul conține o eroare logică. Este important să se detecteze erorile logice. O modalitate de a realiza acest lucru sistematic este prin **testare.**

Testare

Rularea unui program utilizând diferite date de intrare pentru a identifica existența unor eventuale erori logice.

În etapa de testare, tester-ul are nevoie să știe care ar trebui să fie rezultatele obținute. Acestea s-ar putea referi la cerințele din specificațiile programului pentru a verifica funcționarea corectă a programului, conform scopului său inițial.

După remedierea erorilor, programul va fi retestat pentru asigurarea faptului că acel bug a fost cu adevărat remediat.

Testarea conform cerințelor

Regula de bună practică este ca testarea unui program să se facă pe baza unui document conținând specificațiile scrise în etapa de proiectare a programului. Specificațiile oferă claritate cu privire la funcționalitățile programului. Ele conțin cerințele programului.

O specificație pentru un program va conține de cele mai multe ori o listă numerotată a cerințelor. Acesta ar fi un posibil exemplu:

1.3.1: Programul va stinge ecranul monitorului dacă computerul nu este folosit timp de 1 minut.

Un plan de testare ar trebui să includă și un pas de testare a acestei cerințe. Planul ar prevede funcționarea programului pentru 1 minut, timp în care computerul nu este folosit și verificarea închiderii monitorului după 1 minut. Fiecare cerință ar trebui să aibă teste corespunzătoare.

În acest exemplu, dacă după 1 minut programul cedează, atunci există un bug în program. Cu toate acestea, dacă nu se întâmplă nimic după 1 minut, nu se poate spune cu siguranță că programul nu conține erori.

Acest lucru demonstrează faptul că testarea sistematică a conformității cu cerințele programului este extrem de valoroasă. Testarea ar fi identificat o cerință care nu a fost implementată. Testarea conformității cu cerințele ajută de asemenea la identificarea problemelor care apar doar atunci când anumite părți ale programului sunt utilizate. Testarea în acest mod ar putea identifica anumite probele care altfel nu ar fi putut fi găsite.

Identificarea Bug-urilor

Atunci când un program returnează un rezultat greșit sau neașteptat sau dacă programul cedează în mod neașteptat datorită unei erori, nu este întotdeauna evidentă cauza.

Atunci când un program returnează un răspuns greșit, este posibil să nu fie suficientă doar studierea codului pentru identificarea problemei.

O modalitate de a afla mai multe despre bug-ul respectiv este să se includă o serie de diagnosticări suplimentare în cadrul codului. Acestea ar putea consta în instrucțiuni suplimentare print() pentru a afișa valorile variabilelor în diferite puncte ale programului. Aceste instrucțiuni suplimentare de afișare arată modul de modificare al valorilor.

Efortul depus pentru identificarea cauzei unui bug poartă denumirea de **debugging**.

Debugging

Reprezintă rularea unui program defect folosind informații suplimentare de diagnosticare pentru a determina unde anume în cadrul codului există greșeli și pentru a le remedia.

O altă metodă de diagnosticare a unui program este prin rularea lui ‘sub un depanator’. Acesta reprezintă un instrument ce poate fi utilizat pentru rularea unui program linie cu linie. Python conține un debugger ce poate fi activat din fereastra Python shell.

Debugging-ul (depanarea de cod) poate identifica bug-ul și locul primei sale apariții în cadrul programului. Fără debugging, este posibil să nu fie clar unde anume este problema în interiorul codului.

15.4 ÎMBUNĂTĂȚIREA UNUI PROGRAM

Concepte

Odată ce un program este scris, lansat și se bucură de succes, există destul de frecvent presiunea creării unei noi versiuni îmbunătățite a acestuia.

Din punct de vedere comercial, rescrierea completă a programului nu este o idee bună. Un plan mai bun este să se efectueze mici îmbunătățiri care să facă programul mai util pentru mai multe persoane. Astfel de modificări ajută codul să rezolve o problemă mult mai generală decât versiunea inițială.

Capacitatea de utilizare în diverse țări

Dacă vă amintiți, în capitolul despre gândirea computațională, am discutat despre generalizare. Am văzut că design-ul unei mașini de spălat poate fi generalizat astfel încât să fie potrivit pentru mai multe țări. Acest lucru se întâmplă frecvent și în cazul aplicațiilor software.

Aplicațiile pot beneficia de traducerea șirurilor de caractere în alte limbi. De obicei, programatorii încearcă să utilizeze același cod sursă, însă fac codul mai flexibil astfel încât acesta să utilizeze diferite șiruri de caractere pentru diferite limbi, așa cum este indicat printr-un parametru.

Generalizarea codului pentru a-l face potrivit pentru diferite țări poartă uneori numele de **localizare**. Aceasta poate merge dincolo de o simplă traducere. Un program cu rețete poate suporta diferite tipuri de măsurători. De exemplu, poate suporta măsurarea temperaturii în grade Fahrenheit sau Celsius. Acest lucru ar face programul util mai multor țări.

Localizarea poate, de asemenea, să atragă atenția asupra altor oportunități legate de generalizare.

Localizarea privind furnizarea cantităților de ingrediente în diferite unități de măsură, după cum este necesar în diferite țări, ar putea conduce la alte generalizări utile. Un program îmbunătățit ar putea calcula cantitățile din diferite unități. Acesta ar putea permite utilizatorului programului să solicite ajustări ale dimensiunii porției și numărului de porții, programul recalculând cantitățile.

Rezolvarea unei probleme mai mari

În capitolul privind gândirea computațională, am văzut că soluția problemei de organizare a unui festival de muzică ar putea fi generalizată. Experiența în organizarea unui festival de muzică ar putea contribui la rezolvarea unor probleme mai ambițioase legate de organizarea unui turneu muzical.

În căutarea unor oportunități de îmbunătățire a unui program de rețete, micile adăugiri care, de exemplu, numără calorii sau adună informații nutriționale, ar putea generaliza programul de rețete și l-ar putea include într-un program mai amplu pentru sănătate și nutriție.

Îmbunătățirea reușită a unui program trebuie să echilibreze ambiția cu caracterul practic. O schimbare mare a funcționalității unui program poate necesita o rescriere a acestuia sau poate necesita o nouă bibliotecă/modul (pachet) pentru a aduce noi funcționalități. Modificările majore, cum ar fi adăugarea recunoașterii vocale într-un program, depășesc ceea ce se poate obține cu îmbunătățiri incrementale mici.

Prezentarea unui program

Viziunea unui programator asupra unui program este foarte diferită de viziunea asupra programului pe care trebuie să o aibă un utilizator sau un manager. Atunci când prezentați un program, acordați atenție publicului și aspectelor relevante pentru aceștia. Acest lucru este deosebit de important atunci când se comunică îmbunătățirile propuse. Pseudocodul și schemele logice pot fi relevante pentru a demonstra modul de funcționare al unui algoritm. Însă nu comunică scopul și valoarea.

Comunicarea scopului

Atunci când comunicați scopul unui program, explicați ce problemă rezolvă programul. În general, programele sunt utile în anumite aspecte ale unei probleme repetitive. Computerele sunt bune la automatizare. În comunicarea scopului, poate fi util să comunicați de asemenea și ce părți ale problemei *nu* rezolvă programul.

Un program de procesare de text salvează un autor de la tastarea aceluiași text de multe ori pe măsură ce lucrează prin diferite versiuni ale unui document. Un program de procesare de text nu ajută însă foarte mult un autor să găsească cuvintele potrivite și să identifice idei pentru personaje și intrigă.

Comunicarea valorii

Valoarea unui program sau a îmbunătățirilor acestuia depinde de cine îl va folosi pe piață și de concurență. Managerii ar putea dori să știe dacă există deja un program similar sau dacă noul program poate fi ușor copiat de alții. Dacă algoritmii sunt noi, creativi și inovativi, este posibil să le patenți și astfel să le protejați împotriva copierii.

Dacă programul folosește un format de fișier proprietate, care stochează datele într-un mod în care alte programe nu îl pot citi cu ușurință, acest lucru poate împiedica alte companii să-și creeze propria versiune.

În comunicarea valorii unui program, evidențiați ceea ce este unic în privința acestuia. Arătați că aspectele unice sunt de valoare pentru utilizatorii finali. Un program valoros este cel care economisește timpul și banii utilizatorului final. Acolo unde este posibil, susțineți afirmațiile dvs. cu privire la valoarea unui program cu numere.

15.5 EXERCIȚII RECAPITULATIVE

1. Care dintre următoarele expresii conține o eroare de sintaxă?
   a. 2+4+8
   b. (2+4)*8
   c. 2/4/8
   d. 2+(4*8))

2. Un test logic a fost proiectat pentru a verifica dacă variabilele c, d și e sunt puse în ordine, cu c cel mai mic și e cel mai mare. Care dintre următoarele conține o eroare logică?
   a. if (c<d) and (d<e) :
   b. if (e>d) and (d>c) :
   c. if (c#d) and (d#e) :
   d. if (e<d) and (d<c) :

3. Testarea programului reprezintă:
   a. Chestionarea clientului dacă specificațiile reprezintă exact ce își dorește el de la program.
   b. Remedierea unui bug atunci când un utilizator raportează o eroare.
   c. Situația în care o persoană cu mai multă experiență verifică codul sursă al programului și ajută la îmbunătățirea lui.
   d. O abordare specială de identificare a bug-urilor din program prin rularea programului folosind diverse date de intrare.

4. Depanarea (Debugging) reprezintă:
   a. Verificarea conformității programului cu cerințele pentru a vedea dacă acesta funcționează așa cum ar trebui.
   b. Îmbunătățirea programului prin adăugarea de noi caracteristici.
   c. Efortul depus pentru identificarea cauzei unei probleme și remedierea acesteia.
   d. Îmbunătățirea unui program prin eliminarea caracterisiticilor care nu mai sunt necesare.

5. Una dintre cerințele inițiale pentru un anumit program este să solicite numele dvs. și apoi să îl rețină. Care dintre următoarele lucruri v-ați aștepta să le vedeți în cadrul codului pentru a respecta această cerință?
   a. O funcție recursivă
   b. O instrucțiune input()
   c. Un bug
   d. Un cod pentru a converti un șir de caractere într-un număr întreg.

6. O companie de software vrea să facă îmbunătățiri la versiunea 1 a programului său de gestionare a rețetelor. Care dintre următoarele este o modalitate bună de a identifica posibile îmbunătățiri?
   a. Adăugarea unui cod pentru desenarea unei ferigi în mod recursiv.
   b. Utilizarea motorului de căutare Google pentru căutarea expresiei “robots AND cooking”
   c. Căutarea unor mici modificări care ar putea face programul mai util pentru mai multe persoane.

7. În descrierea valorii comerciale a unui nou program, cea mai bună metodă este:
    a. Prezentarea unei scheme logice extrem de detaliate pentru a evidenția complexitatea programului.
    b. Descrierea beneficiilor pentru utilizatori și susținerea acestui lucru cu date numerice, dacă este posibil.
    c. Explicarea faptului că programul este nou și, ca urmare, trebuie să fie de valoare.
    d. Explicarea faptului că programul utilizează doar algoritmi cunoscuți, utilizați la scară largă și extrem de sigur

Editura ECDL ROMANIA

Telefon: 021.316.99.22
Fax: 021.319.72.27
E-mail: editura@ecdl.ro
Website: www.ecdl.ro