Cuprins

1. Instructiuni

1.1. While

Sintaxa while are rolul de a rula un bloc de sintaxe cat timp conditia este adevarata.

```
while conditie:
    sintaxa1
    sintaxa2
    …
    sintaxaN
```

Exemplu:

```
x = 10
while x>1:
    print(“x este”, x)
    x = x  - 1
```

Astfel, dupa rulari succesive in care valoarea variabilei x a luat valori de la 10 la 2, se iese din while pentru ca x = 1, deci conditia de rularea a sintaxelor din while devine falsa.

Urmatorul exemplu:

```
x = 10
while x>1:
```

```
        x = x + 1
        break
```

ne arata existenta unei bucle infinite de rulari avand in vedere ca x >1 este indeplinita intotdeauna.

break are rolul de a implementa o terminare fortata a procesului de rulare.

```
while True:
        euro = input("Valoare euro pentru convertire: ")
        if euro.isdigit():
                euro=int(euro)
                print("Valoarea convertita este: ", euro*4.5, " RON")
        else:
                print("Valoarea nu este un numar!")

        quit = input("Apasa q pentru a iesi si orice tasta pentru a repeta conversia: ")
        if quit.upper() == "Q":
                break
        else:
                pass
```

```
Valoare euro pentru convertire: 3
Valoarea convertita este:  13.5  RON
Apasa q pentru a iesi si orice tasta pentru a repeta conversia: 2
Valoare euro pentru convertire: r
Valoarea nu este un numar!
Apasa q pentru a iesi si orice tasta pentru a repeta conversia: q
```

Dupa cum se poate observa, introducerea unui numar intreg duce la calcularea conversiei in euro. In cazul in care se introduce orice alt caracter in afara de numar intreg, se va afisa pe ecran un mesaj informativ in care se informeaza utilizatorul ca nu a fost introdus un numar. Daca se doreste continuarea, deci introducerea unui alt numar se va apasa orice alta tasta in afara de Q, iar daca se doreste parasirea conversiei, atunci se apasa tasta q. Cu ajutorul lui break se iese din while, in timp ce pass ajuta la iesirea din instructiunea de block IF-ELSE.
In urmatoarele exemple testam rolul lui break, pass and continue,

```
x = 10
while x > 1:
        print("x este {}".format(x))
        pass
```

```
    x-=1
```

```
x este 10
x este 9
x este 8
x este 7
x este 6
x este 5
x este 4
x este 3
x este 2
```

Deci se poate observa ca instructiunea pass nu are niciun rol, blocul de instructiuni urmand sa fie executat in continuare.

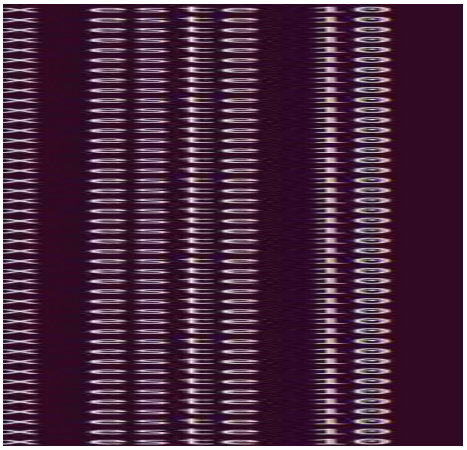
```
x = 10
while x > 1:
        print("x este {}".format(x))
        break
        x-=1
```

```
x este 10
```

Observam ca utilizarea lui break duce la iesirea din blocul de executie al instructiunii while, astfel ca, mesajul print apare o singura data, deci pana la executarea lui break.

```
x = 10
while x > 1:
        print("x este {}".format(x))
        continue
        x-=1
```

Comanda continue are rolul de a face un salt la rularea respectiva fara a intrerupe bucla. Prin urmare va întrerupe rularea blocului de sintaxe pentru rularea dată si se va incepe rularea de la inceput a blocului de expresii de sub while.



1.2. For

Instructiunea for ruleaza de un numar fix de ciclari si are urmatoarea structura:

```
for variabila_temporar in range():
        bloc_expresii
```

sau

```
for iterator in lista:
        bloc_sintaxe
```

```
lista1 = [‘Ana’, ‘are’, ‘mere’]
for x in lista1:
        print(x)
```

Ana
are
mere

Exemple:

```
for i in range(10):
        print("Valoarea lui i este {}".format(i))
```

```
Valoarea lui i este 0
Valoarea lui i este 1
Valoarea lui i este 2
Valoarea lui i este 3
Valoarea lui i este 4
Valoarea lui i este 5
Valoarea lui i este 6
Valoarea lui i este 7
Valoarea lui i este 8
Valoarea lui i este 9
```

```
for i in range(0, 50, 5):
    print(i)
```


```
for var in "abcde":
        print(var)
```

2. Structuri de date

2.1. Siruri de caractere

Sa presupunem că avem o variabilă numita cuvant ce stochează un șir de caractere.
Dorim sa afisam primul caracter din sirul respectiv, numit index. Afisarea se realizeaza prin apelarea:

```
exemplu = "abcde"
print(exemplu[0])
```

Aceaste paranteze patrate aplica principiul indexarii unde fiecare element al sirului are un index. Totusi trebuie sa fim atenti la index. Daca acesta este depasit interpretorul ne va genera eroare.

```
Traceback (most recent call last):
  File "exemplul4.py", line 3, in <module>
    print(exemplu[5])
IndexError: string index out of range
```

Fiecare șir de caractere poate fi prelucrat pe principiul indexării unui șir de caractere. În cazul în care dorim să prelucram dintr-un șir de caractere ultimul caracter, dar nu știm lungimea acelui sir de caractere deoarece este dat de utilizator prin introducere de la tasatatura, Python vine cu o modalitate ușoara de a manipula astfel de șiruri. Se introduce conceptul de indexare inversă. Prin urmare var[-1] va reprezenta ultimul caracter din șirul de caractere stocat de variabila var.

Exemplu:

```
iterator=0
cuvantInvers=""

exemplu = input("Scrie un cuvant:\t")

print("Cuvantul tau este {} \n".format(exemplu))

lungime = len(exemplu)

print("Lungimea cuvantului este {}:".format(lungime))

for var in exemplu:
    print("{} trebuie sa fie egal cu  {}".format(exemplu[iterator], var))
    iterator+=1
    cuvantInvers=var+cuvantInvers

print("\nCuvantul scris invers este: {} ".format(cuvantInvers))

input("\n\nApasa <enter> pt a iesi.")
```

```
Scrie un cuvant:        radio
Cuvantul tau este radio

Lungimea cuvantului este 5:
r trebuie sa fie egal cu  r
a trebuie sa fie egal cu  a
d trebuie sa fie egal cu  d
i trebuie sa fie egal cu  i
o trebuie sa fie egal cu  o

Cuvantul scris invers oidar:


Apasa <enter> pt a iesi.
```

len are rolul de a da lungimea cuvantului

Tipurile de date ce sunt indexabile se impart in mutabile si imutabile. Un șir de caractere este un element imutabil. Presupunând ca avem o variabilă x care susține șirul de caractere “bec”. Dacă am greșit cuvantul și dorim să schimbam prima litera în c, am putea încerca să facem acest lucru cu indexarea șirului de caractere.

```
x = "bec"
print(x[0])
x[0]="c"
```

```
b
Traceback (most recent call last):
  File "exemplu5.py", line 3, in <module>
    x[0]="c"
TypeError: 'str' object does not support item assignment
```

Soluția este reatribuirea unei valori noi acelei variabile. Astfel x=”cec” este soluția problemei modificării șirului de caractere.

```
x = "sir de caractere"
x.replace("r", "c")
print(x)
print(x.replace("r", "c"))
```

```
sir de caractere
sic de cacactece
```

Cu ajutorul metodei find putem găsi șiruri de caractere în interiorul altor șiruri de caractere. Apelarea se face prin variabila.find(“sir de caractere cautat”). Daca nu gaseste sirul cautat returneaza -1. Daca gaseste sirul de caractere returneaza indexul primei litere cautate din sirul de caractere

```
x = "abcdef"
print("Litera cautata este r. Ea exista? {}".format(x.find("r")))
print(x.find("bc"))
print(x.find("de"))
print(x.find("b"))
```

```
Litera cautata este r. Ea exista? -1
x.find('bc') 1
x.find('de') 3
x.find('b') 1
```

Metoda split returnează o lista de elemente împărțite după un șir dat. Lista este un concept încă neînvățat, dar trebuie totuși subliniat ca elimina sirul de caractere dupa ce se face split (de referința).

Apelarea se face prin variabila.split(“sir de caractere ce împarte șirul”)

Aceste metode nu schimba șirul inițial! Prin urmare dacă dorim să prelucram ulterior va trebui să atribuim valoarea returnata de metoda unei variabile.

```
y = "sir de caractere"
print(" {}".format(y.split(" ")))
```

```
y.split('') ['sir', 'de', 'caractere']
```

None

None are scopul de a putea inițializa o variabila, dar fără a susține nici o valoare.

La testarea lui None ca si condiție va fi tratat ca False.

None nu poate fi transformat in nici un alt tip si nu se poate aduna,repeta etc.

2.2. Tuplu

Tuplu reprezintă o serie de secvențe care pot conține numere, șiruri sau alte tuple. Prin urmare poți stoca tot felul de date. Inițializarea unui tuplu gol se face astfel:

```
x=( )
x = (True, False, 4, 5, 6, "sir", ("tuplu nou", 2))
print(x, type(x))
```

```
((True, False, 4, 5, 6, 'sir', ('tuplu nou', 2)), <type 'tuple'>)
```

Un alt element ce trebuie stiut in ceea ce priveste tuplurile este ca se pot concatena.

```
x+(1, 2, 3)
print("x+(1,2,3) => {}".format(x))
x+=(1,2,3)
print("x+=(1, 2, 3) => {}".format(x))
```

```
x+(1,2,3) => (True, False, 4, 5, 6, 'sir', ('tuplu nou', 2))
x+=(1, 2, 3) => (True, False, 4, 5, 6, 'sir', ('tuplu nou', 2), 1, 2, 3)
```

```
inventar = None
if not inventar:
    print("Momentan nu ai niciun produs in inventar.")

inventar = ("paine","rosii","branza","ceapa")

print("\nTuplul inventar este acum : {} \n".format(inventar))

#print "\nElementele inventarului sunt:"
for item, value in enumerate(inventar):
    print("Produs {} => {}".format(item, value))

print("\n\nIn total avem {} produse".format(len(inventar)))
```

input("\n\nApasa <enter> pt a iesi.")

```
Momentan nu ai niciun produs in inventar.

Tuplul inventar este acum : ('paine', 'rosii', 'branza', 'ceapa')

Produs 0 => paine
Produs 1 => rosii
Produs 2 => branza
Produs 3 => ceapa


In total avem 4 produse


Apasa <enter> pt a iesi.
```

2.3. Lista

Lista este un alt tip de variabilă care are toate caracteristicile unui tuplu, exceptând imutabilitatea. Prin urmarea o listă permite rescrierea unui element component, proprietate numita mutabilitate. Acesta calitate ne permite flexibilitatea de care avem nevoie în programare.

Elementele ce formeaza o lista nu au o conditie de unicitate.

O lista este un tip de structura de date care are proprietatea de mutabilitate si indexare.

```
x = [True, False, 4, 5, 6, 7, "sir", ['lista noua'], ('tuplu', 4)]
print('Lista noastra este => {}'.format(x))

print('Testam mutabilitatea unei liste, astfel')
x[2] = 8
print('x[2] era 4, iar dupa aplicarea x[2] = 8 acesta are valoarea =>
{}'.format(x[2]))
```

```
Lista noastra este => [True, False, 4, 5, 6, 7, 'sir', ['lista noua'], ('tuplu', 4)]
Testam mutabilitatea unei liste, astfel
x[2] era 4, iar dupa aplicarea x[2] = 8 acesta are valoarea => 8
```

Utilizam acelasi program de la tupluri, dar schimbam structura de date inventar in lista.

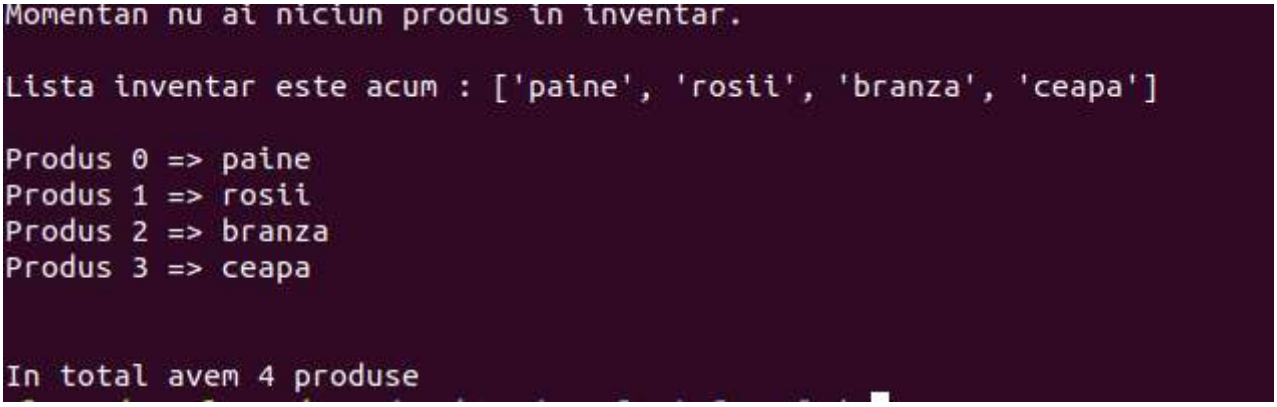
```
inventar = None
if not inventar:
    print("Momentan nu ai niciun produs in inventar.")

inventar = ["paine","rosii","branza","ceapa"]

print("\nLista inventar este acum : {} \n".format(inventar))

#print "\nElementele inventarului sunt:"
for item, value in enumerate(inventar):
    print("Produs {} => {}".format(item, value))

print("\n\nIn total avem {} produse".format(len(inventar)))
```



Similar cu metodele de manipulare ale unui șir de caractere exista metode, diferite ce-i drept, și pentru liste.

Lista - metode de manipulare

Metoda	**Descriere**
append(*value*)	adauga *value* la finalul listei.
sort()	Sorteaza elementele dupa valoarea cea mai mica.
reverse()	Reinverseaza ordinea listei. Primul element devine ultimul, al doilea devine penultimul...

count(*value*)	Indexeaza de cate ori va gasi acel sir de caractere (*value*).Zero inseamna ca nu a gasit nimic, 1 inseamna ca a gasit o potrivire
index(*value*)	Returneaza prima pozitie(indexul) cand *value* apare în lista. Returneaza eroare daca nu este intalnita.
insert(*i*, *value*)	Insereaza *value* la pozitia *i*.
pop([*i*])	Returneaza *value* pozitia *i* și sterge valoarea *value* din lista. E optionala numarul pozitie *i* și poate fi omis ; în acest caz ultimul element va fi sters și returnat.
remove(*value*)	Sterge prima intrare pe care o intalneste cu valoarea *value* din lista.

Lista - secvente imbricate

O lista poate conține si o alta lista. Se poate merge mai departe de atât, astfel ca o lista poate conține o alta lista care conține o alta lista... Acest procedeu se numește secvențe imbricate (imbricat=suprapus parțial) (eng. nested sequence).

Accesarea unui astfel de lista din lista se face prin indexarea repetata.

```
lista=[0, 1, ["al doilea nivel", ["al treilea nivel", "abc"]]]
print("Lungimea listei este => {}".format(lista))
print("Elementul lista[0] este {}".format(lista[0]))
print("Elementul lista[2] este {}".format(lista[2]))
print("Elementul lista[2][0] este {}".format(lista[2][0]))
print("Elementul lista[2][1][1] este {}".format(lista[2][1][1]))
print("Lungimea liste lista[2][1] este {}".format(len(lista[2][1])))
```

```
Lungimea listei este => [0, 1, ['al doilea nivel', ['al treilea nivel', 'abc']]]
Elementul lista[0] este 0
Elementul lista[2] este ['al doilea nivel', ['al treilea nivel', 'abc']]
Elementul lista[2][0] este al doilea nivel
Elementul lista[2][1][1] este abc
Lungimea liste lista[2][1] este 2
```

Daca dorim sa prelucram de la un index pana la sfarsitul unui range nu mai completam valoarea finala. Simiar si pentru prelucrarea primelor elemente ale listei.

abc[::2] va selecta toate valorile din 2 in 2 din sirul de caractere dat. Putem folosi ce pas consideram necesar.

```
print("Pana la finalul liste lista[1:] => {}".format(lista[1:]))
print("De la inceputul liste lista[3:] => {}".format(lista[:3]))
print("Elementele din 2 in 2 => {}".format(lista[::2]))
```

```
Pana la finalul liste lista[1:] => [1, ['al doilea nivel', ['al treilea nivel', 'abc']]]
De la inceputul liste lista[3:] => [0, 1, ['al doilea nivel', ['al treilea nivel', 'abc']]]
Elementele din 2 in 2 => [0, ['al doilea nivel', ['al treilea nivel', 'abc']]]
```

2.4. Set

Daca o lista permite intrări duplicat putem avea probleme in cazul in care dorim sa avem intrări unice, probleme ce pot fi rezolvate dacă utilizam verificări anterioare.
Declararea unui set se face cu ajutorul unor paranteze de tip acolada ce conțin toate elementele cu virgula, similar cu o lista.

```
s = {1, 2, 3, 4, 5}
print("Tipul este => {}".format(type(s)))
x = set()
x.add(1)
x.add(2)
x.add(3)
```

print("Tipul lui x este => {}".format(type(x)))

```
Tipul este => <type 'set'>
Tipul lui x este => <type 'set'>
```

In primul rand trebuie sa reținem ca putem transforma o lista în set și un set în lista folosind funcția set() sau funcția list()
Un set este o structura destul de unica deoarece nu suporta indexarea. Daca nu avem index nu putem avea mutabilitate sau imutabilitate.

print("Setul nu suporta indexarea")
print(x[1])

```
Setul nu suporta indexarea
Traceback (most recent call last):
  File "exemplul10.py", line 10, in <module>
    print(x[1])
TypeError: 'set' object does not support indexing
```

Totusi prin conventie un set este un tip de date mutabil.

Un set nu permite concatenarea intre doua seturi.Nu permite împărțirea sau repetiția cu un integer sau un alt set.

print("x+{6}")

x+{6}

```
x+{6}
Traceback (most recent call last):
  File "exemplul10.py", line 13, in <module>
    x+{6}
TypeError: unsupported operand type(s) for +: 'set' and 'set'
```

Operatii Set

Mai jos regasim un tabel ce explica fiecare concept în parte

Concept	Însemnătate

intersectie	set1 & set2	Ce este comun la doua set
uniune	set1 | set2	Unirea elementelor cu condiţia ca un element sa fie unic în set
diferenta	set1 - set2	Ce este in set1 si nu e in set 2
Diferenta simetrica	set1 ^ set2	Ce este in set1 si nu e in set2 unit cu ce este in set2 si nu e in set1

In programele noastre vom avea nevoie de adaugarea unui element sau de ștergerea unui element dintr-un set. Operatia de adaugare se realizează cu metoda de manipulare set.add(element_nou). Operatia de stergere se realizeaza cu metoda de manipulare set.discard(element_sters)

```
>>> set1
set([1, 2, 3, 4, 5])
>>> set2
set([8, 2, 4, 10, 6])
>>> set1.add(9)
>>> set1
set([1, 2, 3, 4, 5, 9])
>>> set1.discard(9)
>>> set1
set([1, 2, 3, 4, 5])
```

Metoda de manipulare set.discard(element_sters) nu ridica o eroare in cazul in care elementul sters nu exista in set.
De asemenea, asa cum putem utiliza la o lista cuvantul cheie **in** pentru a verifica daca exista un element in lista, cu siguranta putem utilza cuvantul cheie **in** si la un set.

```
>>> set1
set([1, 2, 3, 4, 5])
>>> if 2 in set1:
        print "exista 2"


exista 2
>>> if "2" in set1:
        print "exista '2'"
else:
        print "Nu exista '2'"


Nu exista '2'
```

2.5. Dictionar

Un dicționar este format din perechi de tip “cheie”: „valoare” ; unde cheia și valoarea pot fi de asemenea și numere liste etc. . Deifinirea unui dictionar se face cu ajutorul parantezelor de tip acolada astfel:

Un dictionar este un tip de date care structureaza foarte bine datele.

Dictionar={cheie:val}

Cheia poate fi numerica sau sir de caractere. Nu se accepta alt tip de cheie.

Valoarea poate fi orice tip de date.

```
>>>
>>> a = {1:"abc","2":2}
>>> #nume={key:value}

>>> {[1]:"Salut"}

Traceback (most recent call last):
  File "<pyshell#67>", line 1, in <module>
    {[1]:"Salut"}
TypeError: unhashable type: 'list'
```

Un dictionar este are proprietatea de mutabilitate ca in exemplul de mai jos.

```
>>> x={1:"Salut"}
>>> x
{1: 'Salut'}
>>> x[1]="test"
>>> x
{1: 'test'}
>>> x["abc"]=["a","b","c"]
>>> x["abc_tuplu"]=(["a","b","c"],"d","e")
>>> x["abc_set"]=set(("a","b","c","d","e"))
>>> x
{1: 'test', 'abc_tuplu': (['a', 'b', 'c'], 'd', 'e'), 'abc': ['a', 'b', 'c'], 'a
bc_set': set(['a', 'c', 'b', 'e', 'd'])}
>>>
```

Cheia trebuie sa existe, in caz contrar interpretorul genereaza eroare. In acest caz trebuie sa aflam daca o cheie exista in dictionar. Putem folosi cuvantul cheie in pentru a afla daca exista o cheie in dictionar.

```
>>> d
{1: 1, '2': 10}
>>> if (10 in d):
        print "10 in dictionar"
else:
        print "10 nu e in dictionar"


10 nu e in dictionar
>>> if ("2" in d):
        print "2 in dictionar"
else:
        print "2 nu e in dictionar"


2 in dictionar
```

Asa cum o lista sau un set are metode de manipulare putem avea si metode de manipulare. Mai jos putem vedea un tabel cu toate metodele mai uzuale utilizate pentru manipularea unui dictionar:

Metoda	**Descriere**
get(*key*, [*default*])	Returneaza valoarea cheii *key*.Daca cheia nu este gasita atunci se returneaza cuvantul optional *default*. Daca cheia nu exista și cuvantul *default* nu este specificat, atunci se va returna None.
keys()	Returneaza o lista cu toate cheile din dictionar.
values()	Returneaza o lista cu toate valorile din dictionar.
items()	Returneaza o lista cu toate elementele din dictionar Fiecare element este un tuplu de doua elemnete de tip (cheie,valoare)..
pop(*key*)	Sterge elementul key:value din dictionar daca key exista. In cazul în care nu exista va returna eroare. Returneaza value.
del dictionar[*key*]	Sterge perechea key:value din dictionar daca key exista. In cazul în care nu exista va returna eroare. Nu returneaza nimic.