LAB 3. Liste și dicționare, funcţii cu argumente, funcții built-in. Fişiere.

Obiective:

Studenții trebuie:

Să înțeleagă și să utilizeze **Liste și dicționare** în programele Python.
Să exploreze **funcţii cu argumente, funcții built-in** și să le aplice în rezolvarea problemelor.

I. Liste

Listele sunt folosite pentru a stoca mai multe articole într-o singură variabilă.

Listele sunt unul dintre cele 4 tipuri de date încorporate în Python utilizate pentru a stoca colecții de date, celelalte 3 sunt Tuple , Set și Dictionary , toate cu calități și utilizări diferite.

Listele sunt create folosind paranteze drepte:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

Listează articole

Elementele din listă sunt ordonate, modificabile și permit valori duplicate.

Elementele din listă sunt indexate, primul articol are index [0], al doilea element are index [1]etc.

Elementele din listă sunt indexate și le puteți accesa referindu-vă la numărul de index:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

Ordonat

Când spunem că listele sunt ordonate, înseamnă că articolele au o ordine definită, iar această ordine nu se va schimba.

Dacă adăugați elemente noi la o listă, acestea vor fi plasate la sfârșitul listei.

Notă: Există câteva metode de listă care vor schimba ordinea, dar în general: ordinea articolelor nu se va modifica.

Schimbabil

Lista este modificabilă, ceea ce înseamnă că putem modifica, adăuga și elimina elemente dintr-o listă după ce aceasta a fost creată.

Permite duplicate

Deoarece listele sunt indexate, listele pot avea articole cu aceeași valoare:

```
thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)
```

Lungimea listei

Pentru a determina câte articole are o listă, utilizați len()funcția:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

Lista articole - Tipuri de date

Elementele din listă pot fi de orice tip de date:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

Modificați valoarea articolului

Pentru a modifica valoarea unui anumit articol, consultați numărul de index:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Adăugați articole

Pentru a adăuga un articol la sfârșitul listei, utilizați metoda append() :

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

II. Dictionare

Dicționarele sunt folosite pentru a stoca valorile datelor în perechi cheie:valoare.
Un dicționar este o colecție ordonată*, modificabilă și nu permite duplicate.

Dicționarele sunt scrise cu paranteze și au chei și valori:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Dicţionar Items

Elementele din dicționar sunt ordonate, modificabile și nu permit duplicate.
Elementele din dicționar sunt prezentate în perechi cheie:valoare și pot fi făcute referire folosind numele cheii.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

Duplicate nu sunt permise

Dicționarele nu pot avea două articole cu aceeași cheie:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
```

Dicţionar Length

Pentru a determina câte articole are un dicționar, utilizați len()funcția:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(len(thisdict))
```

Elemente de dicționar - Tipuri de date

Valorile din articolele de dicționar pot fi de orice tip de date:

tip()

Din perspectiva lui Python, dicționarele sunt definite ca obiecte cu tipul de date „dict”:

```
<class 'dict'>
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(type(thisdict))
```

Constructorul dict().

De asemenea, este posibil să utilizați constructorul dict() pentru a realiza un dicționar

```
thisdict = dict(name = "John", age = 36, country = "Norway")
print(thisdict)
```

Accesarea articolelor

Puteți accesa articolele unui dicționar referindu-vă la numele cheii acestuia, între paranteze drepte:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
print(x)
```

Există, de asemenea, o metodă numită get()care vă va oferi același rezultat:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.get("model")
print(x)
```

Obțineți cheile

Metoda keys()va returna o listă cu toate cheile din dicționar.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = thisdict.keys()

print(x)
```

Schimbați Valorile

Puteți modifica valoarea unui anumit articol, referindu-vă la numele cheii acestuia:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

thisdict["year"] = 2018

print(thisdict)
```

Actualizați dicționarul

Metoda update()va actualiza dicționarul cu elementele din argumentul dat.

Argumentul trebuie să fie un dicționar sau un obiect iterabil cu perechi cheie:valoare.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})

print(thisdict)
```

Adăugarea de articole

Adăugarea unui articol în dicționar se face prin utilizarea unei noi chei de index și atribuirea unei valori:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Eliminarea articolelor

Există mai multe metode de a elimina articole dintr-un dicționar:

Metoda pop()elimină elementul cu numele cheii specificat:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

Cuvântul delcheie elimină elementul cu numele cheii specificat:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

Metoda clear()golește dicționarul:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

III. Funcții Python

O funcție este un bloc de cod care rulează numai atunci când este apelată.

Puteți trece date, cunoscute ca parametri, într-o funcție.

O funcție poate returna date ca rezultat.

Crearea unei funcții

În Python, o funcție este definită folosind cuvântul cheie `def` :

```
def my_function():
  print("Hello from a function")
```

Apelarea unei funcții

Pentru a apela o funcție, utilizați numele funcției urmat de paranteze:

```
def my_function():
  print("Hello from a function")

my_function()
```

Argumente

Informațiile pot fi transmise în funcții ca argumente.

Argumentele sunt specificate după numele funcției, în interiorul parantezei. Puteți adăuga câte argumente doriți, doar separați-le cu o virgulă.

Următorul exemplu are o funcție cu un singur argument (fname). Când funcția este apelată, transmitem un prenume, care este folosit în interiorul funcției pentru a tipări numele complet:

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Argumentele sunt adesea scurtate la *args* în documentațiile Python.

Parametri sau argumente?

Termenii *parametru* și *argument* pot fi folosiți pentru același lucru: informații care sunt transmise într-o funcție.

Din perspectiva unei funcții:

Un parametru este variabila listată între paranteze în definiția funcției.

Un argument este valoarea care este trimisă funcției atunci când este apelată.

Numărul de argumente

În mod implicit, o funcție trebuie apelată cu numărul corect de argumente. Înseamnă că, dacă funcția ta așteaptă 2 argumente, trebuie să apelezi funcția cu 2 argumente, nu mai multe și nu mai puțin.

Exemplu

Această funcție așteaptă 2 argumente și primește 2 argumente:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

Dacă încercați să apelați funcția cu 1 sau 3 argumente, veți obține o eroare:

Exemplu

Această funcție așteaptă 2 argumente, dar primește doar 1:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil")
```

Argumentele cuvintelor cheie

De asemenea, puteți trimite argumente cu sintaxa *cheie* = *valoare* .

În acest fel ordinea argumentelor nu contează.

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Valoarea implicită a parametrului

Următorul exemplu arată cum se utilizează o valoare implicită a parametrului.

Dacă apelăm funcția fără argument, aceasta folosește valoarea implicită:

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Trecerea unei liste ca argument

Puteți trimite orice tip de date de argument unei funcții (șir, număr, listă, dicționar etc.) și va fi tratat ca același tip de date în interiorul funcției.

De exemplu, dacă trimiteți o Listă ca argument, aceasta va fi totuși o Listă când ajunge la funcția:

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

Valori returnate

Pentru a permite unei funcții să returneze o valoare, utilizați `return` instrucțiunea:

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Declarația de trecere

`function`definițiile nu pot fi goale, dar dacă dintr-un motiv oarecare aveți o `function`definiție fără conținut, introduceți `pass`declarația pentru a evita o eroare.

```
def myfunction():
  pass
```

Argumente numai pentru cuvinte cheie

Pentru a specifica că o funcție poate avea doar argumente de cuvinte cheie, adăugați `*,` *înaintea* argumentelor:

```
def my_function(*, x):
  print(x)

my_function(x = 3)
```

Fără aceasta, `*,` aveți voie să utilizați argumente poziționale chiar dacă funcția așteaptă argumente ale cuvintelor cheie:

```
def my_function(x):
  print(x)

my_function(3)
```

Recursiune

Python acceptă, de asemenea, recursiunea funcției, ceea ce înseamnă că o funcție definită se poate autodenomina.

Recursiunea este un concept matematic și de programare comun. Înseamnă că o funcție se numește singură. Acest lucru are avantajul de a însemna că puteți parcurge datele pentru a ajunge la un rezultat.

Dezvoltatorul ar trebui să fie foarte atent cu recursiunea, deoarece poate fi destul de ușor să introduci în scrierea unei funcții care nu se termină niciodată, sau una care utilizează cantități în exces de memorie sau puterea procesorului. Cu toate acestea, atunci când este scrisă corect recursiunea poate fi o abordare foarte eficientă și elegantă din punct de vedere matematic pentru programare.

În acest exemplu, `tri_recursion()` este o funcție pe care am definit-o să se numească singură ("recurse"). Folosim variabila `k` ca date, care scade (`-1`) de fiecare dată când recurgem. Recursiunea se termină când condiția nu este mai mare decât 0 (adică când este 0).

Pentru un nou dezvoltator, poate dura ceva timp pentru a afla cum funcționează exact acest lucru, cel mai bun mod de a afla este testarea și modificarea acestuia.

```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("Recursion Example Results:")
tri_recursion(6)
```

Mai jos sunt 5 probleme pentru fiecare nivel de complexitate. Alegeti o problema din fiecare nivel

---

Nivel Ușor

1. Scrie o funcție `numere_pare(lista)` care primește o listă de numere și returnează doar cele pare.
2. Scrie o funcție `suma_pozitive(lista)` care returnează suma tuturor numerelor pozitive din listă.
3. Scrie o funcție `lungimi_cuvinte(lista)` care primește o listă de cuvinte și returnează o listă cu lungimile acestora.
4. Scrie o funcție `elimina_duplicate(lista)` care returnează o listă fără elemente duplicate.
5. Scrie o funcție `produs_impare(lista)` care returnează produsul tuturor numerelor impare dintr-o listă.

Nivel Mediu

1. Modifică `numere_pare(lista)` astfel încât să returneze și numărul total de numere pare găsite.
2. Modifică `suma_pozitive(lista)` astfel încât să returneze și media aritmetică a numerelor pozitive.
3. Modifică `lungimi_cuvinte(lista)` pentru a returna doar cuvintele mai lungi de 5 caractere.
4. Modifică `elimina_duplicate(lista)` astfel încât să returneze și numărul de duplicate eliminate.
5. Modifică `produs_impare(lista)` astfel încât să gestioneze cazul în care nu există numere impare (returnează un mesaj corespunzător).

Nivel Dificil

1. Extinde `numere_pare(lista)` pentru a returna și suma numerelor pare, nu doar lista lor și numărul lor.
2. Extinde `suma_pozitive(lista)` astfel încât să returneze o listă nouă care conține doar numerele pozitive, împreună cu suma și media lor.
3. Extinde `lungimi_cuvinte(lista)` astfel încât să returneze un dicționar în care cheia este cuvântul, iar valoarea este lungimea sa.
4. Extinde `elimina_duplicate(lista)` astfel încât să returneze un dicționar în care cheia este elementul original, iar valoarea este numărul de apariții ale acestuia.

5. Extinde `produs_impare(lista)` astfel încât să returneze și lista numerelor impare folosite în calcul.