

# Python Session 10

---



# Today

---

- HW from last course
- Functions
- Exception handling
- File manipulation
- Homework

# Scopes

---

O variabila este accesibila regiunii in care a fost creata

global

---

Variabilele definite la nivel global.

Incluzand variabilele definite in interiorul unor functii, dar specificate ca fiind variabile globale

---

local

Variabilele definite intr-o functie

---

nested

Variabilele definite intr-o functie nested in alta functie

# Arbitrary Arguments

---

unknown number of arguments

- Syntax

\*arguments

- Rezultat

Tuple - tip de date secential, o lista cu elemente, pot fi accesate pe baza indexului

- Definirea unui parametru

Folosind asterix in momentul in care declarăm un parametru.

Definim un parametru cu '\*' in momentul in care nu stim cate argumente vor fi folosite in momentul apelarii functiei.

- Dacă noi stim că folosim un număr minim de argumente, trebuie să realizăm ca valorile argumentelor sunt pasate parametrilor pe baza pozitiei.

- Cand apelam o functie care accepta 2 argumente normale si un arbitrary argument, trebuie sa oferim cel putin 2 argumente functiei, orice argument in plus va face parte din parametrul definit cu asterix.

# Arbitrary Arguments

---

```
def my_function(*kids):  
    print("The youngest child is " + kids[2])
```

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

# Keyword Arguments

specific arg value to parameter

- Syntax

key = value

nume = valoare

- In momentul apelarii, cand oferim argumente, putem ca si argumente sa specificam numele si valoarea parametrului, in modul acesta, ordinea argumentelor nu mai conteaza.
- ```
def my_function(child3, child2, child1):
    print("The youngest child is " + child3)
```

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

# Arbitrary Keyword Arguments

---

unknown number of arguments

- Syntax

\*\*kwargs

- Rezultat

Dictionary

- Cand apelam functia care contine \*\*kwargs cu key arguments, se creeaza in spate un dictionar care este pasat functiei, unde cheia este numele parametrului folosit si valoarea este valoarea asociata.

```
def my_function(**kid):
    print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

# Recursivitate

---

- Python accepta recursivitate, insemnand ca permite functiei sa apeleze pe ea însăși
- Exemplu:

- Sum of first n natural numbers:

```
def sumnums(n):
    if n == 1:
        return 1
    return n + sumnums(n - 1)
```

```
print(sumnums(3))
print(sumnums(6))
print(sumnums(9))
```

- Exercitiu : Scrie o functie care sa iti calculeze factorialul unui numar folosind recursivitatea

# Exception Handling

- Exceptions = Errors
- Exceptions in Python

In momentul in care Python intampina o problema in programul executat, va ridica o exceptie(eroare).

- Un exception este un obiect cu descrierea problemei si stack traceback(raportul executiei - cine ce a apelat si in ce ordine)

# Exception Handling

- **KeyError**
- **NameError**  
Labels(variables, functions, etc)
- **SyntaxError**  
Sintaxa incorecta
- **IndexError**  
Apelarea unui index inexistent
- **ZeroDivisionError**  
Impartirea la 0

# Exception Handling

- **TypeError**

Cand unul din elementele unei operatii nu este tipul potrivit.

int + str

int + list

etc

# try except

- Pentru a evita terminarea executiei unui program trebuie sa gestionam o posibile erori
- try except
- try
  - Codul executat care este monitorizat in cazul unei erori
- except:
  - codul executat in cazul unei erori in blocul de cod 'try'

# try except else finally

- try
- except
- else

Codul executat in cazul in care codul din 'try' a fost executat cu succes fara ca o eroare sa fie ridicata

- finally

Codul executat la finalul statementului try except indiferent daca au fost erori gasite sau nu.

# Specific Exceptions

---

- Putem insera cate expresii except dorim in functie de tipul de eroare
- Folosind built-in exceptions putem decide in functie de exception ce cod ruleaza in except, si in continuare
- In cazul in care nu stim tipul exceptiei, dar vrem mesajul putem apela obiectul 'Exception'  
except Exception as variabila:  
`print(variabila)`

# Specific Exceptions

---

```
import logging
logger = logging.getLogger(__name__)

try:
    a= 2/0
except ZeroDivisionError as variable:
    logger.exception(variable)
except KeyError:
    print("Key error")
except (IndexError,SyntaxError) as error:
    print("INDEX ERROR OR SYNTAX ERROR")
```

# raise

- În cazul în care suntem în ramura except, putem ridica acea eroare (throw - raise) cum face și Python în mod normal
- Syntax:

```
except:  
    print('am intampinat o eroare')  
    raise
```

# Exercitiu

Folosind try except

Fara verificarea tipului de date

Indiferent de problemele intampinate, programul nu trebuie sa ridice o eroare!

- **Sa scriem o functie care primeste 2 argumente**

Daca argumentele pot fi impartite punem rezultatul intr-o variabila si mentionam ca impartirea a fost facuta cu succes

Daca nu pot fi impartite, incercam sa le concatenam, mentionand ca am facut concatenarea cu succes si rezultatul il punem intr-o variabila

Daca nu pot fi impartite si nici concatenate, printam un mesaj ca avem nevoie de elemente de acelasi tip.

La finalul statementului try printam un mesaj care sa contina rezultatul

- **Sa apelam functia cu:**

1, 1

'unu', 1

'unu', 'doi'

# File manipulation

---

- built-in function `open()`

Cu ajutorul functiei `open()` putem manipula fisiere.

- `close()` method

Este de preferat dupa ce deschidem un fisier pentru a fi editat, sa fie inchis.

`close()` method este apelata asupra fisierului deschis cu `open()`

- Syntax:

```
file = open('file_name', 'mode')  
file.close()
```

- File

Pentru a manipula un fisier specific, trebuie sa mentionam calea catre acel fisier.

Daca calea nu este specificata, ci doar numele fisierului, Python cauta acel fisier in folderul in care este fisierul python executat.

# File manipulation modes

---

- Modul “w”

Mode: “w” - modul write, in cazul in care fisierul nu exista, va fi creat

In cazul in care fisierul exista, orice text adaugat va rescrie tot fisierul.

```
file = open('file_name', 'w')
```

Atentie: modul 'w' va rescrie fisierul.

- **write()** method

Cu ajutorul metodei write() apelata asupra fisierului deschis putem adauga informatii prin argument.

```
file.write('text')
```

# File manipulation modes

---

- Modul “a”

Mode: “a” - modul append, in cazul in care fisierul nu exista, va fi creat.

In cazul in care fisierul exista, vom porni in modul append si orice informatie adaugata este scrisa la sfarsitul fisierului.

```
file = open('file_name', 'a')
file.write('test')
file.close()
```

- Modul “r”

Mode: “r” - modul read, in cazul in care fisierul nu exista, vom avea o eroare care mentioneaza ca fisierul nu exista.

In cazul in care fisierul exista, vom porni in modul read si putem citi informatiile din fisier.

```
file = open('file_name', 'r')
file.close()
```

# File manipulation

---

- **read() method**

Cu ajutorul metodei `read()` apelata asupra fisierului deschis, putem citi continutul fisierului  
`file.read()`

- **read() arguments**

Putem oferi un argument metodei `read()`, sa specificam cate caractere vrem sa citim de la  
inceputul fisierului

`file.read(5)`

- **ATENTIE!**

In momentul in care apelam `read()`, python itereaza prin caracterele fisierului, si in cazul in care  
am citit primele 5 caractere, rezultatul urmatorului apel `read()` o sa inceapa de la ultimul  
caracter  
citat

# File manipulation

---

- **readline()** method

Va returna prima linie din fisier, in cazul in care apelam din nou metoda, vom primi a2a linie.

- **readlines()** method

Va returna o lista cu elemente separate prin newline.

- Iterating through files:

Prin modul 'r' putem citi si putem itera prin fiecare linie din fisier.

```
for line in file:
```

```
    print(line)
```

# File removal

---

- Pentru a sterge fisiere in Python avem nevoie de modulul OS(operating system):

Considerand ca Python poate rula pe aproape orice sistem de operare, eliminarea fisierelor poate fi diferita de la un os la altul.

- **import os**

Modul in care avem acces la modulul 'os'.

Pentru a folosi metodele, sau functiile din modului os trebuie sa le apelam ca orice alta metoda, folosind punctul asupra obiectului.

- **remove() method**

!Atentie: remove() sterge fisierele de tot! Nu le mai putem gasi in recycle bin!

```
os.remove('calea_catre_fisier_')
```

# Homework

---

- Complete the feedback form (it's anonymous don't worry)
- Add options to the quiz game so the user can start playing or see the high scores.
- After the user plays save the score in a new line in the scores file