Formal Languages and Compilers

Lab2

Introduction to Python

Python -Operators-

Arithmetic operators

Operator	Name	Example x=10 y =3
+	Addition	x+y =13
-	Subtraction	x-y =7
*	Multiplication	x*y =30
/	Division	x/y =3.3333
%	Modulus	x%y =1
**	Exponentiation	x**y =1000
//	Floor division	x//y =3

Assignment operators

Operator	Example	Meaning
=	x=10	x=10
+=	x+=3	x=x+3 =>x=13
-=	x-=3	x=x-3 => x=7
*=	x*=3	x=x*3 => x=30
/=	x/=3	x=x/3 => x=3.3333
%=	x%=3	x=x%3 => x=1
**=	x**=3	x=x**3 => x=1000
//=	x//=3	x=x//3 => x=3

Python -Operators-

Comparison operators

Operator	Name
==	Equal
!=	Not equal
>	Greater than
>	Less than
>=	Grater than or equal to
<=	Less than or equal to

Python -Operators-

Logical operators

Operator	Meaning	Example x=7
and	Returns True if both statements are true	x>2 and x<8 True x>15 and x <20 False x>10 and x <5 False
or	Returns True if one of the statements is true	x>2 or x<8 True x>15 or x <20 True x>10 or x <5 False
not	Reverse the result (if the result is True, it returns False)	not(x>2 and x<8) False not(x>15 and x <20) True not(x>10 and x <5) True not(x>2 or x<8) False not(x>15 or x <20) False not(x>10 or x <5) True

Python -Operators-

Identity operators

Operator	Meaning	Example

is	Returns True if both variables are the same object	x = ["Student", "Professor"]
		y = ["Student", "Professor"]
		z=x
		
		x is y False (same content)
		x is z True (same object)

is not	Returns True if both variables are not the same object	x = ["Student", "Professor"]
		y = ["Student", "Professor"]
		z=x
		
		x is y True
		x is z False

Python -Operators-

Membership operators

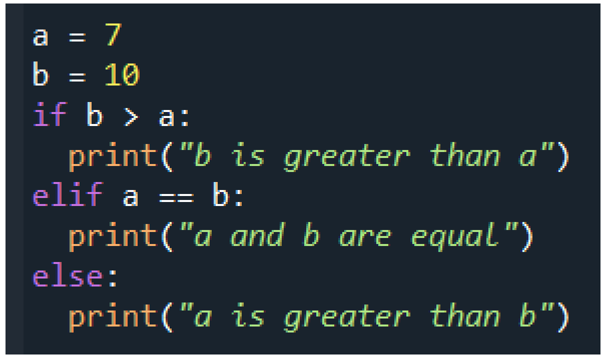
Operator	Meaning	Example

in	Returns True if a sequence with the specified value is present in the object	x = ["Student", "Professor”]
		y=“Student”
		z=“student”
		
		y in x True
		z in x False

not in	Returns True if a sequence with the specified value is not present in the object	x = ["Student", "Professor”]
		y=“Student”
		z=“student”
		
		y not in x False
		z not in x True

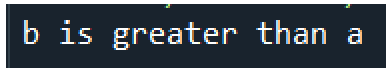
Python
-if elif else-

- **if** statement -> do something
- **elif** -> if the previous condition is not true, try this one
- **else** -> catches all the cases that were not stated in the previous condition
- After an if, elif or else statement, “:” must be put
- Simple example



```python
a = 7
b = 10
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

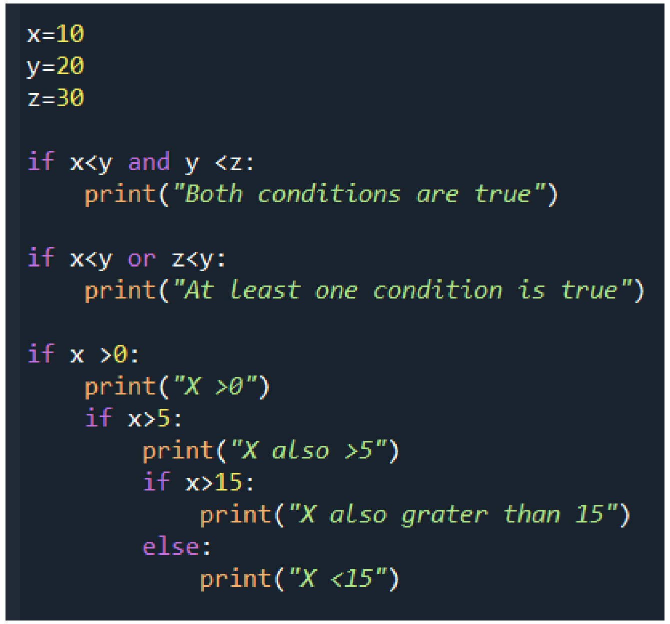
- Output:



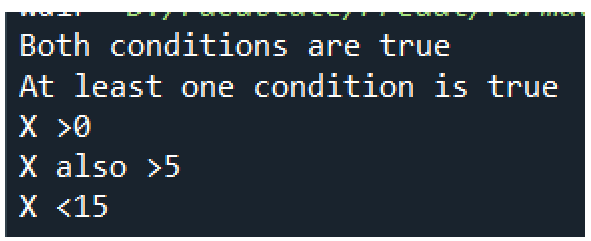
```
b is greater than a
```

Python
-if elif else-

- **Indentation is essential** because it is used to define block of code . Errors will be raised if indentation is not used for statements
- Logical operators such as “and” and “or” can be used to define the conditions
- **Nested If:** if statement inside if statement



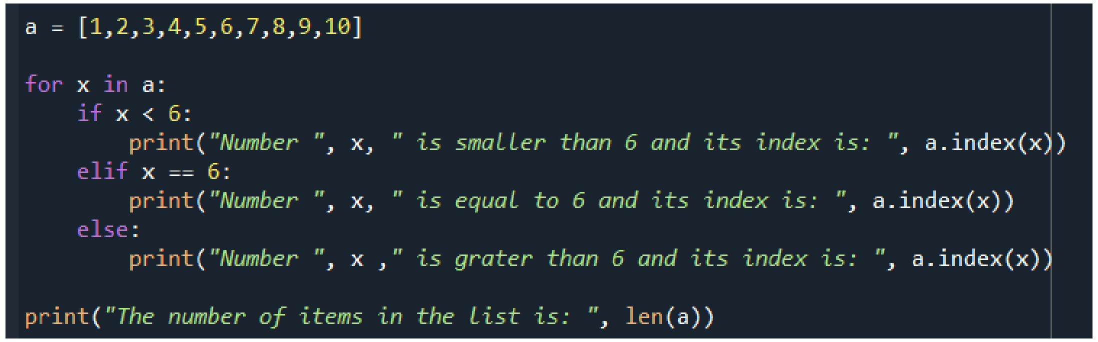
```python
x=10
y=20
z=30

if x<y and y <z:
    print("Both conditions are true")

if x<y or z<y:
    print("At least one condition is true")

if x >0:
    print("X >0")
    if x>5:
        print("X also >5")
        if x>15:
            print("X also grater than 15")
        else:
            print("X <15")
```



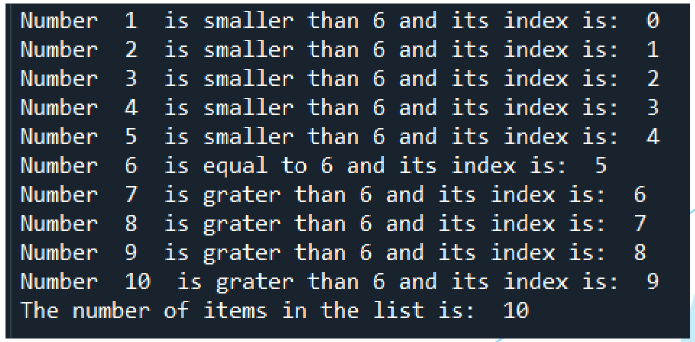
```
Both conditions are true
At least one condition is true
X >0
X also >5
X <15
```

Python
-for loop-

- Used for iterating over a sequence (list, tuple, dictionary, set, string, range)
- We can use it to execute a statement for each element in a sequence
- **break**: stop the loop before making all iterations
- **continue**: stop the current iteration, but continue with the next
- Simple example



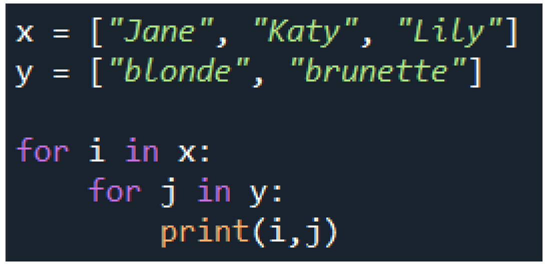
```python
a = [1,2,3,4,5,6,7,8,9,10]

for x in a:
    if x < 6:
        print("Number ", x, " is smaller than 6 and its index is: ", a.index(x))
    elif x == 6:
        print("Number ", x, " is equal to 6 and its index is: ", a.index(x))
    else:
        print("Number ", x ," is grater than 6 and its index is: ", a.index(x))

print("The number of items in the list is: ", len(a))
```



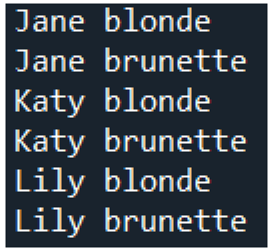
```
Number  1  is smaller than 6 and its index is:  0
Number  2  is smaller than 6 and its index is:  1
Number  3  is smaller than 6 and its index is:  2
Number  4  is smaller than 6 and its index is:  3
Number  5  is smaller than 6 and its index is:  4
Number  6  is equal to 6 and its index is:  5
Number  7  is grater than 6 and its index is:  6
Number  8  is grater than 6 and its index is:  7
Number  9  is grater than 6 and its index is:  8
Number  10  is grater than 6 and its index is:  9
The number of items in the list is:  10
```

Python
-for loop-

- **Nested loop:** loop inside a loop (the inner loop will be executed one time for each iteration of the outer loop)

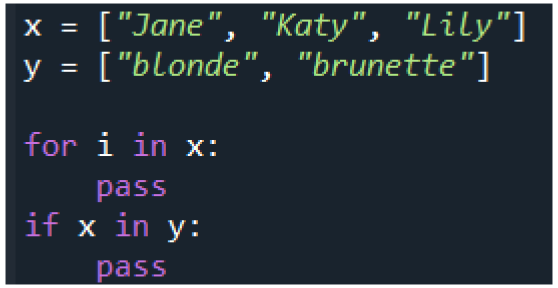


```python
x = ["Jane", "Katy", "Lily"]
y = ["blonde", "brunette"]

for i in x:
    for j in y:
        print(i,j)
```



```
Jane blonde
Jane brunette
Katy blonde
Katy brunette
Lily blonde
Lily brunette
```

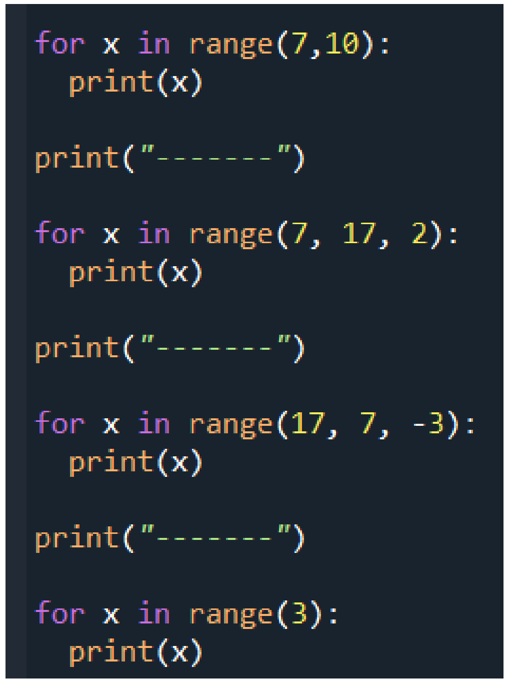
- **pass:** can be used inside an empty loop/if in order to avoid errors



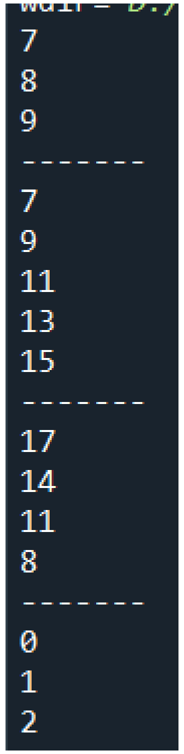
```python
x = ["Jane", "Katy", "Lily"]
y = ["blonde", "brunette"]

for i in x:
    pass
if x in y:
    pass
```

Python
-for loop-

- range() returns a sequence of numbers incremented by 1 (by default)
- range(start, stop, step) : start of the sequence, end of the sequence (the last value is not taken), incrementation (if we need it not to be 1)
- Example with range()



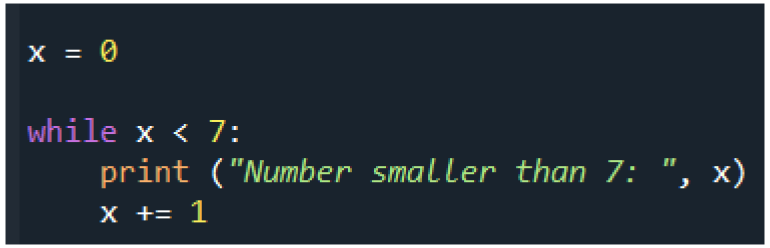
```
for x in range(7,10):
  print(x)

print("-------")

for x in range(7, 17, 2):
  print(x)

print("-------")

for x in range(17, 7, -3):
  print(x)

print("-------")

for x in range(3):
  print(x)
```



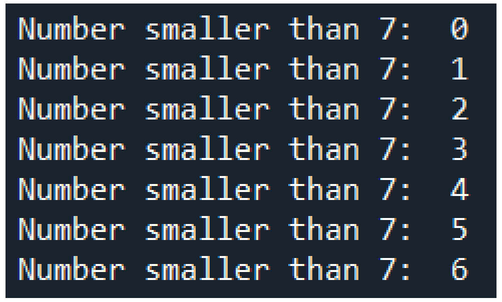
```
7
8
9
-------
7
9
11
13
15
-------
17
14
11
8
-------
0
1
2
```

Python
-while loop-

- Executes a set of statements as long as the condition is true
- Simple example



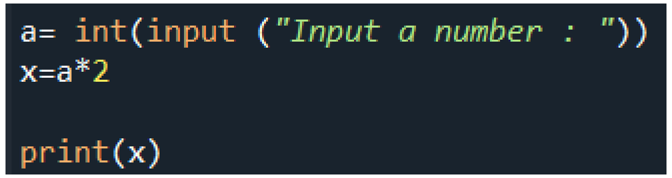
```python
x = 0

while x < 7:
    print ("Number smaller than 7: ", x)
    x += 1
```



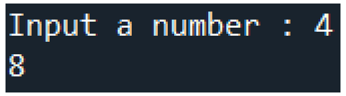
```
Number smaller than 7:  0
Number smaller than 7:  1
Number smaller than 7:  2
Number smaller than 7:  3
Number smaller than 7:  4
Number smaller than 7:  5
Number smaller than 7:  6
```

Python
-User input-

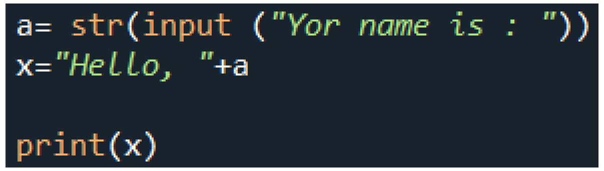
- Python allows user input
- Method: input()
- Is good to use casts (especially when it comes to numbers to avoid errors)
- Simple example



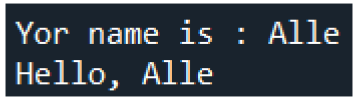
```python
a= int(input ("Input a number : "))
x=a*2

print(x)
```



```
Input a number : 4
8
```



```python
a= str(input ("Yor name is : "))
x="Hello, "+a

print(x)
```



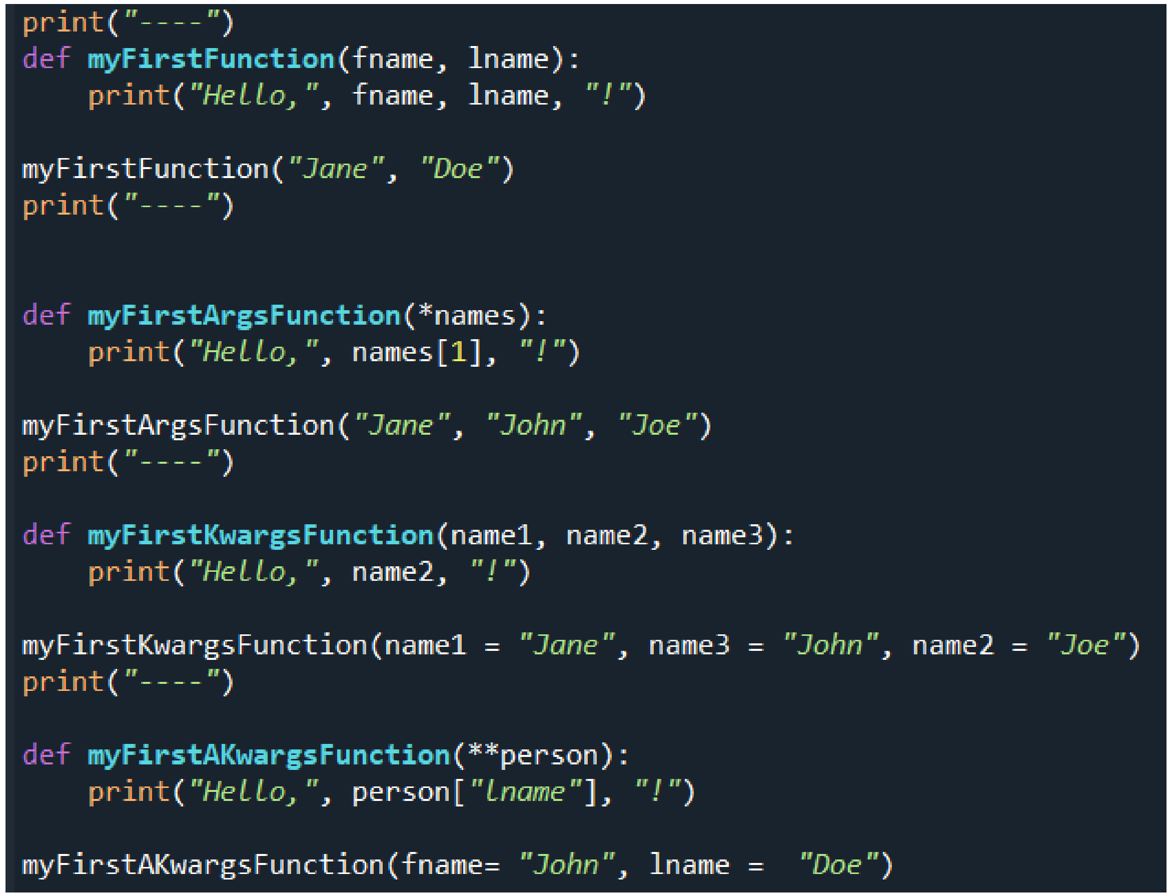
```
Yor name is : Alle
Hello, Alle
```

Python
-Functions-

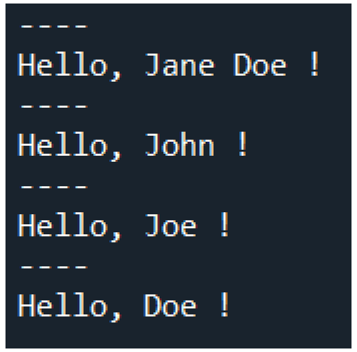
- Function = block of code that runs only when is called
- Defined using “def”: **def nameOfFunction():**
- Arguments: **def nameOfFunction(arg1, arg2, arg3, …):**
- To return a value, “**return**” statement should be used at the end of the function
- Runs only when called: **nameOfFunction()** (the number of arguments used to call a function must be the same with the number of arguments used in defining the function)
- Keywords arguments: **key=value syntax**
- Arbitrary arguments ***args** : used when you do not know how many arguments will be passed
- Arbitrary keyword arguments ****kwargs** : used when you do not know how many keyword arguments will be passed
- pass: used if we have an empty function to avoid errors

Python -Functions-

- Simple examples



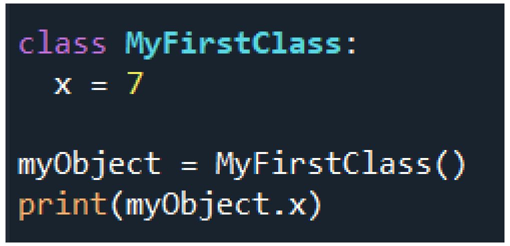
```python
print("----")
def myFirstFunction(fname, lname):
    print("Hello,", fname, lname, "!")

myFirstFunction("Jane", "Doe")
print("----")


def myFirstArgsFunction(*names):
    print("Hello,", names[1], "!")

myFirstArgsFunction("Jane", "John", "Joe")
print("----")

def myFirstKwargsFunction(name1, name2, name3):
    print("Hello,", name2, "!")

myFirstKwargsFunction(name1 = "Jane", name3 = "John", name2 = "Joe")
print("----")

def myFirstAKwargsFunction(**person):
    print("Hello,", person["lname"], "!")

myFirstAKwargsFunction(fname= "John", lname =  "Doe")
```



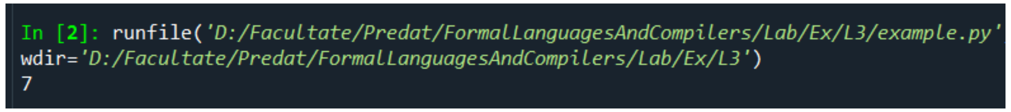
```
----
Hello, Jane Doe !
----
Hello, John !
----
Hello, Joe !
----
Hello, Doe !
```

Python
-Classes-

- Python is OOP (almost everything in Python is an object with its properties and methods)
- A class is an object constructor; it can have multiple functions in it
- Keyword “class” is used to create classes
- Simple example: myFirstClass is a class. myObject is an object of myFirstClass used to print the value of x
- pass: used if we have an empty class to avoid errors



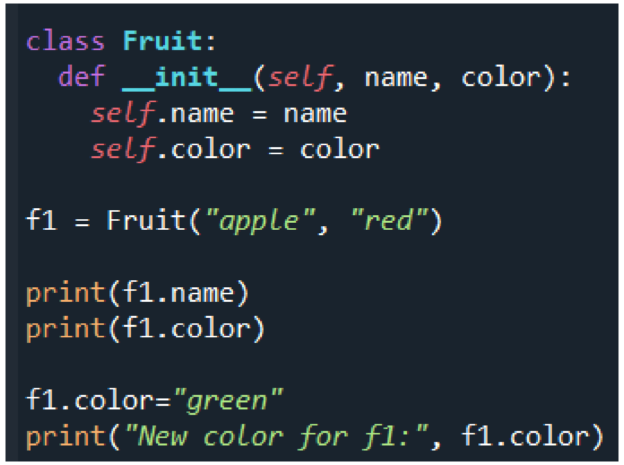
```python
class MyFirstClass:
  x = 7

myObject = MyFirstClass()
print(myObject.x)
```



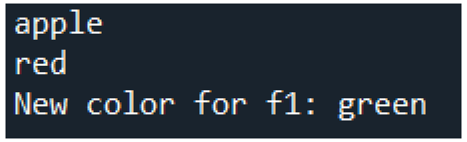
```
In [2]: runfile('D:/Facultate/Predat/FormalLanguagesAndCompilers/Lab/Ex/L3/example.py'
wdir='D:/Facultate/Predat/FormalLanguagesAndCompilers/Lab/Ex/L3')
7
```

Python
-Classes-

- __init__() : function which assigns values to the data members of the class when an object of class is created
- self : parameter, reference to the current instance of a class, used to access variables from that class (it can be named differently, but used with the same scope)
- You can modify properties of an object easily
- del : used to delete object properties or objects
- Simple example:



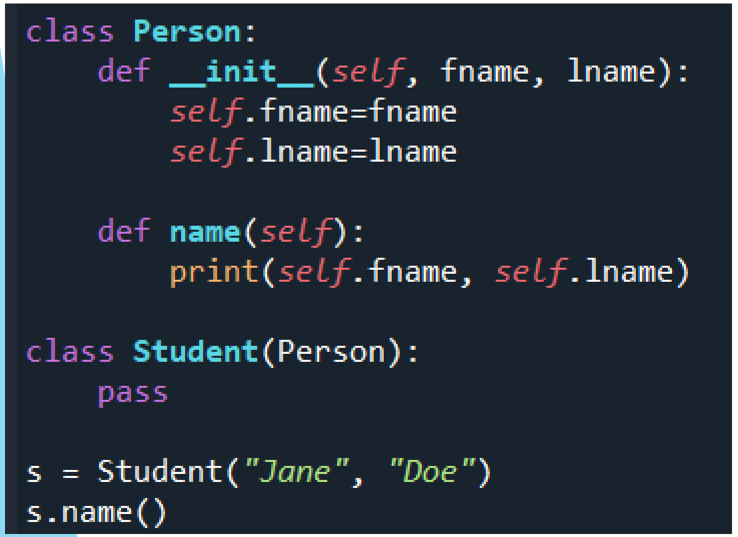
```python
class Fruit:
  def __init__(self, name, color):
    self.name = name
    self.color = color

f1 = Fruit("apple", "red")

print(f1.name)
print(f1.color)

f1.color="green"
print("New color for f1:", f1.color)
```



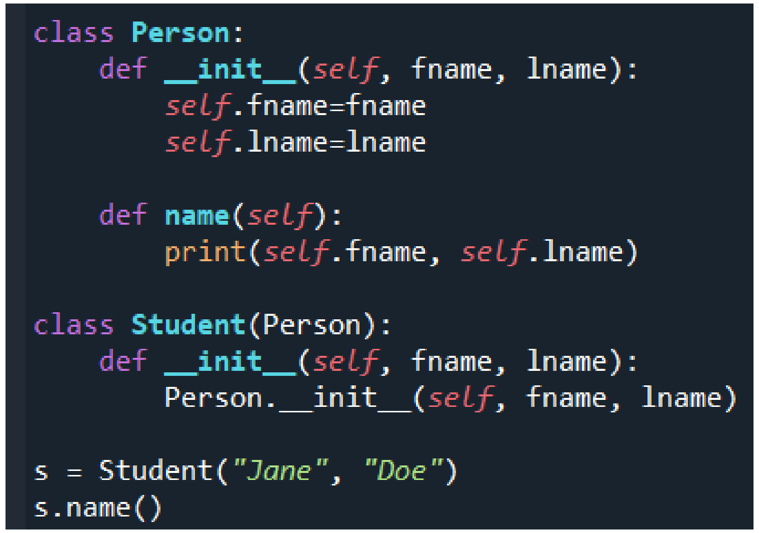
```
apple
red
New color for f1: green
```

Python
-Classes-

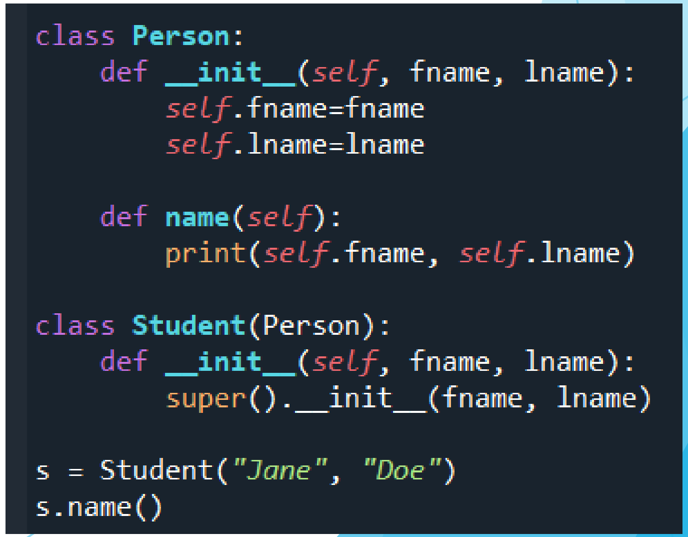
- Inheritance: define a class (child) that inherits all the methods and properties of another class (parent)
- If a child class defines an __init__ function it will no longer inherit the parent's properties => we need to call the parent's __init__() function or use super() (function that will make the child inherit all the parent's properties and methods)
- Simple example



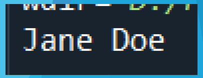
```python
class Person:
    def __init__(self, fname, lname):
        self.fname=fname
        self.lname=lname

    def name(self):
        print(self.fname, self.lname)

class Student(Person):
    pass

s = Student("Jane", "Doe")
s.name()
```



```python
class Person:
    def __init__(self, fname, lname):
        self.fname=fname
        self.lname=lname

    def name(self):
        print(self.fname, self.lname)

class Student(Person):
    def __init__(self, fname, lname):
        Person.__init__(self, fname, lname)

s = Student("Jane", "Doe")
s.name()
```



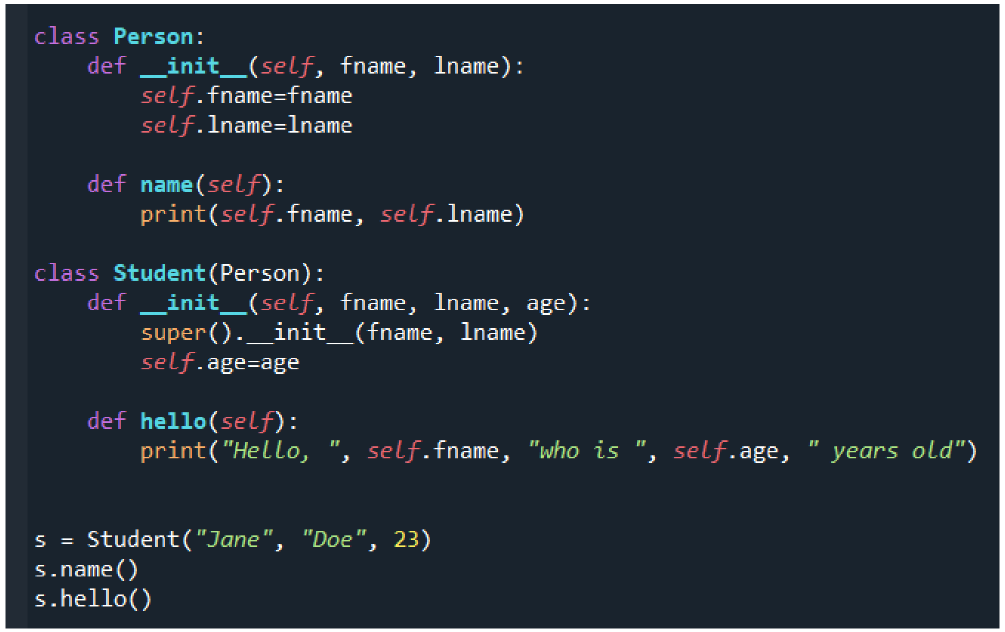
```python
class Person:
    def __init__(self, fname, lname):
        self.fname=fname
        self.lname=lname

    def name(self):
        print(self.fname, self.lname)

class Student(Person):
    def __init__(self, fname, lname):
        super().__init__(fname, lname)

s = Student("Jane", "Doe")
s.name()
```



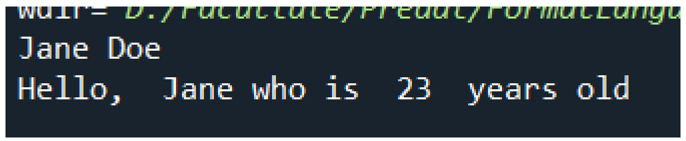
```
Jane Doe
```

Python
-Classes-

- Properties can be added to the child's __init__() function
- Methods must be added to another function (NOT in the one with the parent's name because the parent's inheritance will be overridden)
- Simple example



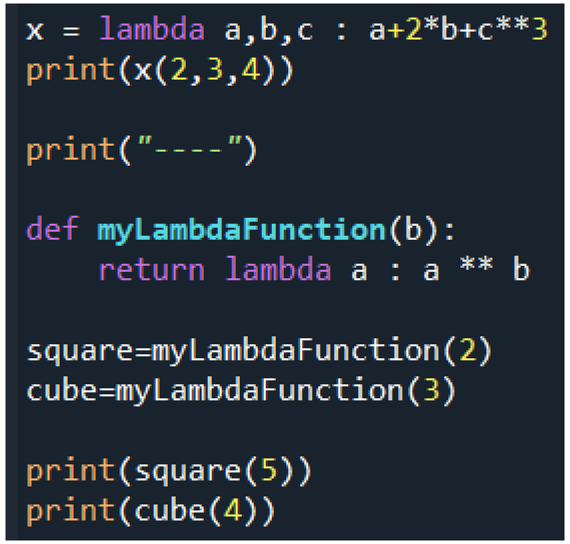
```python
class Person:
    def __init__(self, fname, lname):
        self.fname=fname
        self.lname=lname

    def name(self):
        print(self.fname, self.lname)

class Student(Person):
    def __init__(self, fname, lname, age):
        super().__init__(fname, lname)
        self.age=age

    def hello(self):
        print("Hello, ", self.fname, "who is ", self.age, " years old")


s = Student("Jane", "Doe", 23)
s.name()
s.hello()
```



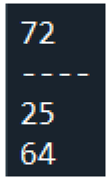
```
wdir= D:/Facultate/Predat/FormalLangu
Jane Doe
Hello,  Jane who is  23  years old
```

Python -lambda-

- Small anonymous function
- Can have multiple arguments, but only one expression: *lambda arguments : expression*
- Mostly used inside another function
- Simple example



```
x = lambda a,b,c : a+2*b+c**3
print(x(2,3,4))

print("----")

def myLambdaFunction(b):
    return lambda a : a ** b

square=myLambdaFunction(2)
cube=myLambdaFunction(3)

print(square(5))
print(cube(4))
```



```
72
----
25
64
```

Exercises

1. Write a program which prints the numbers between 40 and 70 which are divisible by 3
2. Write a program which takes as user input their first and last names and who prints them in reverse order, with a space between
3. Write a program which takes as user input an integer and displays the sum of all numbers from 1 to it [e.g., if the user input is 5, the output will be 15 (1+2+3+4+5)]
4. Write a program which displays ”Success” while looping through the first 10 numbers, starting with 5.
5. Write a program which displays the number of times a letter (count only m,l,c,a,e) appears in the string “Welcome to the lab!” (Hint: use count())
6. Find the factorial of a number inputted by the user
7. Write a program which takes as user input an integer. If the integer is greater than 100, divided by 2 and add 20. If the integer is lower than 100, multiply it by 3 and subtract 200
8. Write a program which takes the user’s input as numbers separated by “,” and transforms that sequence into a list and a tuple (Hint: use split())
9. Write a program which prints the square of all numbers from 1 to a given one

Homework

1. Write a function that prints the Fibonacci series based on the number of terms inputted by the user (e.g., if the user inputs 5, the output should be 0 1 1 2 3)
2. Write a function that computes the great common divisor of two positive numbers.
3. Write a function that computes the least common multiple of two positive numbers.
4. Write a function that takes as input a list of integers and returns 2 lists: one with the even numbers from the initial list and on with the odd numbers (e.g., list1=[1,2,3,4,5,6], the lists obtained using the function will be listEven=[2,4,6] and listOdd=[1,3,5])
5. Write a program which has a class called “Cube” constructed by the length of one side and 3 functions which will compute: the area of one surface, the area of all surfaces and the volume of the cube.
6. Write a program using lambda that will get the power of a specified number. (e.g., power of 2/3 of the specified number 5 will output 25/125)