Lesson 5b: Subprograms

Procedures

Exercise

Write a function called **`alert`** that takes an integer as a parameter and returns an integer.

The function should write a message that says "`Alert: XX seconds remain`" where XX is the integer passed as a parameter. In case the parameter is equal to 0, the message must be "`Alert: Time is over`". The function will always return a 0.

Write the main program that asks the user to enter an integer (number of seconds) and performs a countdown. Each time a second is discounted, the Alert function must be called.

Use:

- `from time import sleep`
- the `sleep(t)` function, where `t` is the number of seconds the execution will be suspended

To define the function remember that we must set:

- The name of the function
- The parameters needed
- The return value

Solution

```
from time import sleep


def alert (sec):
    message = "Alert: "
    if (sec!=0):
        message += str(sec) + " seconds remain"
    else:
        message += "Time is over"

    print(message)
    return 0

# Main program
seconds = int(input("Enter time: "))

for t in range(seconds,-1,-1):
    res = alert(t)
    sleep(1)
```

```
# Main program
seconds = int(input("Enter time: "))

for t in range(seconds,-1,-1):
    alert(t)
    sleep(1)
```

We do not use the returned value in any of the cases

Subprograms

In general

Functions → Sub-algorithms that return a value

Procedures → Sub-algorithms that do not return any value

In Python

There is only the concept of function:

Functions → With **return** at the end of the function.

Procedures → **Without return** at the end of the function.
In this cases, Python returns a special value `None` for us.

Important: `None` is not a string.
It is a value from a special type `NoneType`

Scope of a variable

The scope, or visibility, of a variable indicates in which parts of the program this variable is active, and therefore it can be accessed and modified.

In previous examples, all functions only used variables defined in the function body. These variables are created in the computer memory when the function is called, and they are destroyed when the function finishes.

Scope of variable `y`

```
def f(x):
    x = 1
    x += 1
    print(x,y)
y = 5
f(y)
print(y)
```

Scope of variable `x`

Variables defined in a module (in the example the variable **y** of the main program) are visible in any sub-module called from this module (in this example, the function `f`).

Scope of variables

Each function has its own scope of variables:

```
def f(y):
    x = 1
    x += 1
    print(x)
x = 5
f(x)
print(x)
```

The variables have the same name, but they belong to different scopes

```
Output:
  2
  5
```

Within a function, we can access variables defined outside the function:

```
def g(y):
    print(x)
    print(x + 1)
x = 5
g(x)
print(x)
```

`x` is not defined inside `g`

`g` takes the scope from where the call was made

```
Output:
  5
  6
  5
```

Within a function, **we can NOT modify variables defined outside the function**; we could do it using global variables, but it is not a good practice.

```
def h(y):
    x += 1
x = 5
h(x)
print(x)
```

`x` is not defined within `h`

`h` takes the scope from where the call was made but ...

```
Output:
  File "<ipython-input-166-03cdff1cdb9d>", line 4, in <module>
    h(x)

  File "<ipython-input-166-03cdff1cdb9d>", line 2, in h
    x += 1

UnboundLocalError: local variable 'x' referenced before assignment
```

Scope of a function – Example 1

Each function also has its scope of execution

```
def h():
    x = "abc"
    print("h: x =", x)

def g(x):
    x = x + 1
    print("g: x =", x)
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of
Main Program

h	Code
g	Code
x	3
z	

Scope of a function – Example 1

Each function also has its scope of execution

```
def h():
    x = "abc"
    print("h: x =", x)

def g(x):
    x = x + 1
    print("g: x =", x)
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of Main Program

h	Code
g	Code
x	3
z	

Scope of function `g`

x	3

Scope of a function – Example 1

Each function also has its scope of execution

```
def h():
    x = "abc"
    print("h: x =", x)

def g(x):
    x = x + 1
    print("g: x =", x)
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of Main Program

h	Code
g	Code
x	3
z	

Scope of function `g`

x	4

Scope of a function – Example 1

Each function also has its scope of execution

```
def h():
    x = "abc"
    print("h: x =", x)

def g(x):
    x = x + 1
    print("g: x =", x)
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of Main Program

h	Code
g	Code
x	3
z	

Scope of function `g`

x	4

Scope of a function – Example 1

Each function also has its scope of execution

```
def h():
    x = "abc"
    print("h: x =", x)

def g(x):
    x = x + 1
    print("g: x =", x)
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of
Main Program

h	Code
g	Code
x	3
z	4

Scope of a function – Example 1

Each function also has its scope of execution

```
def h():
    x = "abc"
    print("h: x =", x)

def g(x):
    x = x + 1
    print("g: x =", x)
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of Main Program

h	Code
g	Code
x	3
z	4

Scope of function `h`

x	“abc”

Scope of a function – Example 1

Each function also has its scope of execution

```
def h():
    x = "abc"
    print("h: x =", x)

def g(x):
    x = x + 1
    print("g: x =", x)
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of Main Program

h	Code
g	Code
x	3
z	4

Scope of function `h`

x	“abc”

None

Scope of a function – Example 1

Each function also has its scope of execution

```
def h():
    x = "abc"
    print("h: x =", x)

def g(x):
    x = x + 1
    print("g: x =", x)
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of
Main Program

h	Code
g	Code
x	3
z	4

Scope of a function – Example 2

Each function also has its scope of execution

```
def g(x):
    def h():
        x = "abc"
        print("h: x =", x)
    x = x + 1
    print("g: x =", x)
    h()
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of
Main Program

g	Code
x	3
z	

Scope of a function – Example 2

Each function also has its scope of execution

```
def g(x):
    def h():
        x = "abc"
        print("h: x =", x)
    x = x + 1
    print("g: x =", x)
    h()
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of Main Program

g	Code
x	3
z	

Scope of function `g`

h	Code
x	3

Scope of a function – Example 2

Each function also has its scope of execution

```
def g(x):
    def h():
        x = "abc"
        print("h: x =", x)
    x = x + 1
    print("g: x =", x)
    h()
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of Main Program

g	Code
x	3
z	

Scope of function `g`

h	Code
x	4

Scope of a function

Each function also has its scope of execution

```
def g(x):
    def h():
        x = "abc"
        print("h: x =", x)
    x = x + 1
    print("g: x =", x)
    h()
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of Main Program

g	Code
x	3
z	

Scope of function `g`

h	Code
x	4

Scope of function `h`

x	“abc”

Scope of a function – Example 2

Each function also has its scope of execution

```
def g(x):
    def h():
        x = "abc"
        print("h: x =", x)
    x = x + 1
    print("g: x =", x)
    h()
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of Main Program

g	Code
x	3
z	

Scope of function `g`

h	Code
x	4

Scope of function `h`

x	“abc”

`None`

Scope of a function – Example 2

Each function also has its scope of execution

```
def g(x):
    def h():
        x = "abc"
        print("h: x =", x)
    x = x + 1
    print("g: x =", x)
    h()
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of Main Program

g	Code
x	3
z	

Scope of function `g`

h	Code
x	4

Scope of a function – Example 2

Each function also has its scope of execution

```
def g(x):
    def h():
        x = "abc"
        print("h: x =", x)
    x = x + 1
    print("g: x =", x)
    h()
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of
Main Program

g	Code
x	3
z	4

Scope of a function – Example 2

Each function also has its scope of execution

```
def g(x):
    def h():
        x = "abc"
        print("h: x =", x)
    x = x + 1
    print("g: x =", x)
    h()
    return x

x = 3
z = g(x)
h()
print("main: x =", x)
print("main: z =", z)
```

Scope of
Main Program

g	Code
x	3
z	4

```
Output
NameError: name 'h' is not defined
```

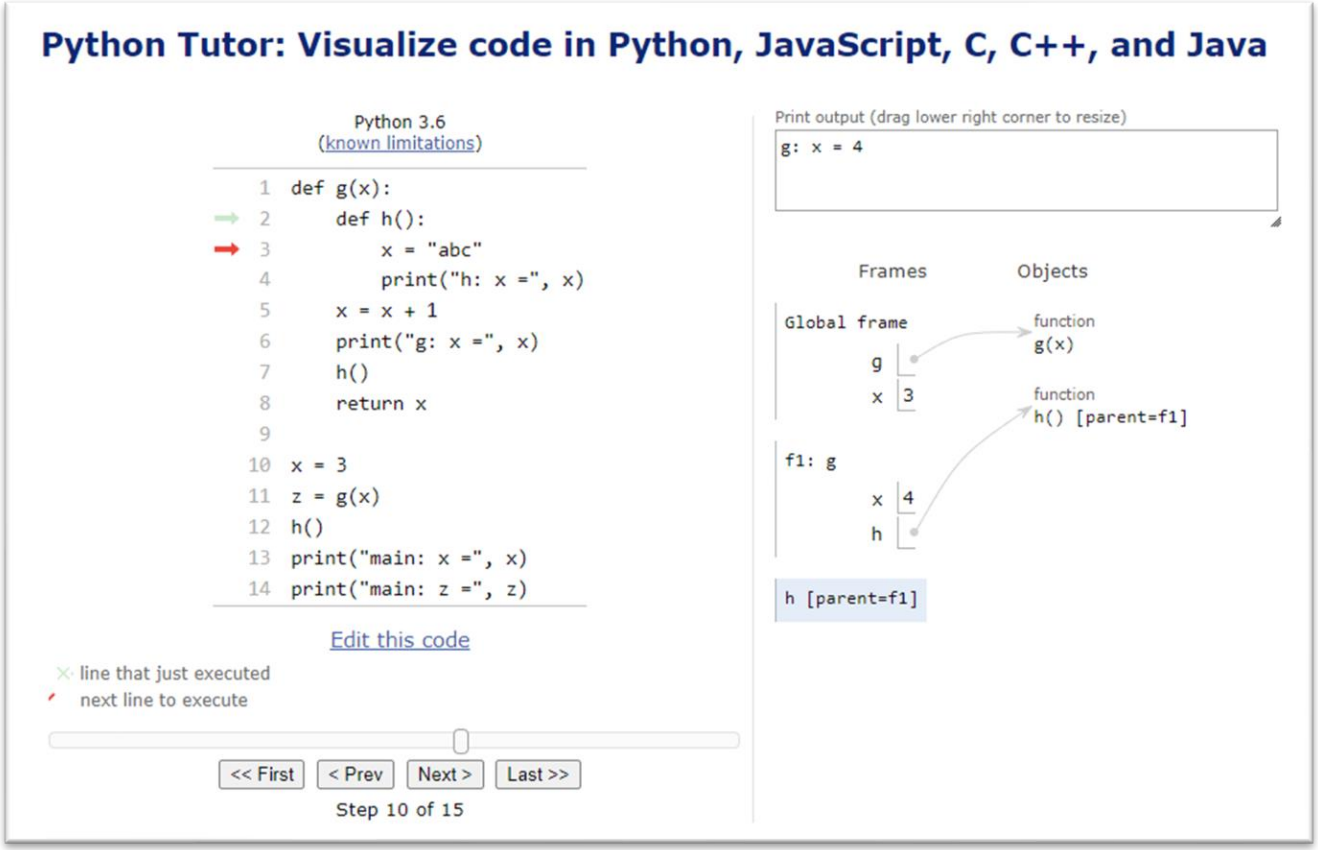
Function `h()` is not defined in the current scope (scope of main program).

The scope applied to functions allows us to implement the concept of **function visibility** (public - private).

Scope of a variable or function

In case you need to see step by step the process of calls and visibility of a variable or function, you can do it in the following website:

http://www.pythontutor.com/



Exercise

Write a function called **`factorial`**, which takes an integer as a parameter and returns the factorial of the number given as a parameter.

Write a function called **`summation`**, which takes an integer as a parameter and returns the summation of the number given as a parameter.

Write a program that allows us to know the time it took to run each of the two functions.

Use:

- `from time import time`
- function `time()` returns the time in seconds as a float number.

Solution

```
from time import time

def factorial(value):
    fact = 1
    for n in range(value,1,-1):
        fact *= n
    return fact

def summation(value):
    sum = 1
    for n in range(value,1,-1):
        sum += n
    return sum

t0 = time()
f=factorial(500)
t1 = time()
sec="seconds"
print("Factorial: %.10f seconds, with result= %d" % (t1-t0,f))


t0 = time()
s=summation(500)
t1 = time()
text = "Summation: {0:.10f} seconds, with result= {1:d}". format(t1-t0,s)
print(text)
```

Print a float with 10 decimals, just here

Formatting

.d	Integer
.f	Float
.s	String

Functions as parameters of other functions

In the example about calculating the execution time, we have repeated exactly the same code, with the only difference of the function evaluated (factorial or summation).

This suggests making a module that receives a function and returns a float:

We are applying the **abstraction principle**. We know:

- the types of the inputs (function and function parameters),
- what the module does (calculates the time it takes for a function to run)
- the output type (float).

Therefore, we do not need to know anything else to be able to use it.

Definition vs. Call

Definition. Keyword **def** followed by the function name (in the example: `timer`) and the formal parameter (in the example: `fnc`, and `arg`) in parentheses.

```
# Timer computes the execution time of fnc
# fnc: Name of the function passed as parameter
# arg: Parameter needed by function fnc

def timer(fnc, arg):
    t0 = time()
    fnc(arg)
    t1 = time()
    return t1-t0
```

Name and formal parameter

Function body

Return

Call. We invoke the function by name (in the example, `timer`) followed by its argument (actual parameter) in parentheses.

call

```
print("Execution time: ", timer(factorial,50), "seconds")
print("Execution time: ", timer(summation,50), "seconds")
```

Function name Actual parameter

Definition vs. Call

Definition. Keyword **def** followed by the function name (in the example: `timer`) and the formal parameter (in the example: `fnc`, and `arg`) in parentheses.

```
# Timer computes the execution time of fnc
# fnc: Name of the function passed as parameter
# arg: Parameter needed by function fnc

def timer(fnc, arg):
    t0 = time()
    fnc(arg)
    t1 = time()
    return t1-t0
```

Name and formal parameter

Function body

Return

Option 1:

```
print("Execution time: ", timer(factorial,50), "seconds")
print("Execution time: ", timer(summation,50), "seconds")
```

Option 2:

```
print("Execution time: %.10f seconds" % timer(factorial,50))
print("Execution time: %.10f seconds" % timer(summation,50))
```

Print a float with 10 decimals