LEARNING TO CODE

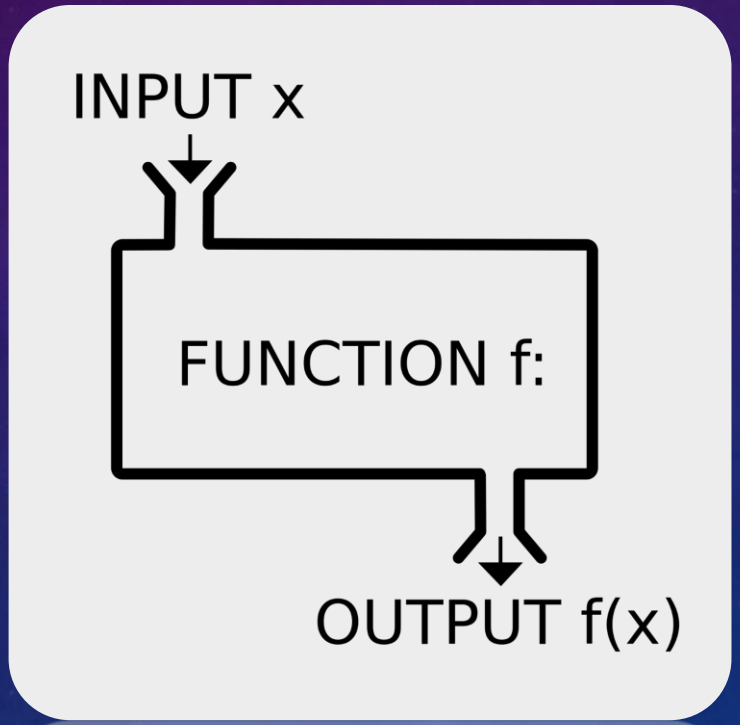
python™

Coordinator: Konstantinos Emmanouilidis

IEEE SB OF THRACE

2017-2018

FUNCTIONS(ΣΥΝΑΡΤΗΣΕΙΣ)



FUNCTIONS

- Divide the code into blocks
- Makes it more readable
- Reusable code
- Save time
- Share code between programmers

BLOCK FORMAT

- A block is an area of code written in the format of:

block_head :

1st block line

2nd block line

3rd block line

.

.

.

- Block_head has the following format:

block_keyword block_name (argument1,argument2,…)

- Block lines are more Python code (even another block)

FUNCTION DEFINITION

Functions use the block format:

- block_keyword: *def*
- Block_name: *function's name*
- arguments : *parameters*
- Block lines are the body of the function.

```
def function_name(param1,param2,param3):
    command1
    command2
    command3
    .
    .
    .
```

EXAMPLES

```
def sum_two_numbers(a,b):
    return a + b
```

```
def my_function():
    print("Hello from my function!")
```

ARGUMENTS OF A FUNCTION

If you specify a default value for one or more arguments then the function can be called with fewer arguments than it is defined to allow.

```
def ask_ok(prompt, retries=4, reminder='Please try again!'):
    while True:
        ok = input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise ValueError('invalid user response')
        print(reminder)
```

Different ways of calling the function:

- with only the mandatory argument:
ask_ok('Do you really want to quit?')
- with one of the optional arguments:
ask_ok('OK to overwrite the file?', 2)
- or even with all arguments:
ask_ok('OK to overwrite the file?', 2, 'Come on, only yes or no!')

ARGUMENTS OF A FUNCTION

- Object reference:

if a mutable object is passed, the caller will see any changes the callee makes to it.

- The default value is evaluated only once.
- This makes a difference when the default is a mutable object such as a list or a dictionary.

```
def f(a, L=[]):
    L.append(a)
    return L
```

```
print(f(1))
print(f(2))
print(f(3))
```

This will print:

```
[1]
[1, 2]
[1, 2, 3]
```

ARGUMENTS OF A FUNCTION

If you don’t want the default to

be shared between subsequent calls,

you can write the function like this instead:

```
def f(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L
```

2ND WAY OF INSERTING ARGUMENTS

- Using keyword arguments of the form *variable=value*.

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
    print("-- This parrot wouldn't", action, end=' ')
    print("if you put", voltage, "volts through it.")
    print("-- Lovely plumage, the", type)
    print("-- It's", state, "!")
```

```
parrot(1000)                                          # 1 positional argument
parrot(voltage=1000)                                  # 1 keyword argument
parrot(voltage=1000000, action='VOOOOOM')             # 2 keyword arguments
parrot(action='VOOOOOM', voltage=1000000)             # 2 keyword arguments
parrot('a million', 'bereft of life', 'jump')         # 3 positional arguments
parrot('a thousand', state='pushing up the daisies')  # 1 positional, 1 keyword
```

Invalid Call of Function:

```
parrot()                     # required argument missing
parrot(voltage=5.0, 'dead')  # non-keyword argument after a keyword argument
parrot(110, voltage=220)     # duplicate value for the same argument
parrot(actor='John Cleese')  # unknown keyword argument
```

HOW TO CALL A FUNCTION??

Write the function's name

followed by (), placing any required arguments within the brackets.

script.py

```
1   # Define our 3 functions
2   def my_function():
3       print("Hello From My Function!")
4
5   def my_function_with_args(username, greeting):
6       print("Hello, %s , From My Function!, I wish you %s"%
    (username, greeting))
7
8   def sum_two_numbers(a, b):
9       return a + b
10
11  # print(a simple greeting)
12  my_function()
13
14  #prints - "Hello, John Doe, From My Function!, I wish you a
    great year!"
15  my_function_with_args("John Doe", "a great year!")
```

RETURN OF A FUNCTION

- The *return* statement returns a value from the function.
- The statement *return* without an expression argument returns *None*.
- Functions without a *return* statement return a value which is called None.

```
>>> def fib(n):    # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print(a, end=' ')
...         a, b = b, a+b
...     print()
...
>>> # Now call the function we just defined:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

This will print:

```
>>> fib(0)
>>> print(fib(0))
None
```