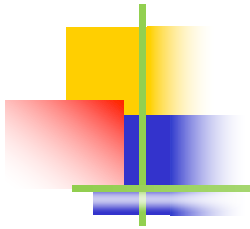

deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Fundamentos de Programação

António J. R. Neves

João Rodrigues

Departamento de Electrónica, Telecomunicações e Informática

Universidade de Aveiro

an@ua.pt / jmr@ua.pt

http://elearning.ua.pt/

Summary

- Functions

Functions

- So far, we have only been using the functions that come with Python, but it is also possible to add new functions.
- A function definition specifies the name of a new function and the sequence of statements that execute when the function is called.
- The first line of the function definition is called the header; the rest is called the body.
- The header has to end with a colon and the body has to be indented.
- `def` is a keyword that indicates that what follows is a function definition.
- The rules for function names are the same as for variable names.

Example


```
def print_hello():
  print("Hello!")
def repeat_hello():
  print_hello()
  print_hello()
#calling the function
repeat_hello()
```

- This example contains two function definitions: `print_hello` and `repeat_hello`. The statements inside the function do not get executed until the function is called.
- A function must be created before being executed. In other words, the function definition has to be executed before the first time it is called.

Flow of execution

- In order to ensure that a function is defined before its first use, you have to know the order in which statements are executed, which is called the flow of execution.
- Execution always begins at the first statement of the program. Statements are executed one at a time, in order from top to bottom.
- Function definitions do not alter the flow of execution of the program, but remember that statements inside the function are not executed until the function is called.
- A function call is like a detour in the flow of execution. Instead of going to the next statement, the flow jumps to the body of the function, executes all the statements there, and then comes back to pick up where it left off.

Parameters and arguments

- Some of the built-in functions we have seen require arguments. For example, when you call `math.sin` you pass a number as an argument.
- Some functions take more than one argument: `math.pow` takes two, the base and the exponent.
- Inside the function, the arguments are assigned to variables called parameters.

```
def repeat_print(msg):
    print(msg)
    print(msg)
```

- This function assigns the argument to a parameter named `msg`. When the function is called, it prints the value of the parameter (whatever it is).

return values

- When you create a variable inside a function, it is local, which means that it only exists inside the function. Parameters are also local.
- Some of the functions we are using, such as the `math` functions, produces results. Other functions, like `print`, perform an action but don't return a value. They are called void functions.
- The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.
- The parameters (arguments) in the Python language becomes another name for the same value that we pass to the function. Confusing? We will explain during the class.

Global vs. Local variables

- Variables that are defined inside a function body have a local scope, and those defined outside have a global.

```
total = 0 # This is a global variable
# Function definition is here
def sum( arg1, arg2 ):
   total = arg1 + arg2 # Here total is local variable
   print("Inside the function local total : ", total)
   return total

# Now you can call sum function
print(sum( 10, 20 ))
print("Outside the function global total : ", total)
```

Keyword and Default arguments

- When you use **keyword** arguments in a function call, the caller identifies the arguments by the parameter name.

```
def printinfo( name, age ):
   print("Name: ", name)
   print("Age ", age)
printinfo( age=50, name="miki" )
```

- A **default** argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

```
def printinfo( name, age = 35 ):
   print("Name: ", name)
   print("Age ", age)
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```

Variable-length arguments

- You may need to process a function for more arguments than you specified while defining the function.
- These arguments are called variable-length arguments and are not named in the function definition.

```
def printinfo( arg1, *vartuple ):
    print(arg1)
    for var in vartuple:
        print(var)
printinfo( 10 )
printinfo( 70, 60, 50 ) #the last two are passed as a tuple
```

- An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call.

Anonymous Functions

- Not declared in the standard manner by using the `def` keyword.
- You can use the `lambda` keyword to create small anonymous functions.

```
# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2
# Now you can call sum as a function
print(”Total: ", sum( 10, 20 )) #Total: 30
```

- Lambda forms can take any number of arguments but return just one value in the form of an expression.
- They cannot contain commands or multiple expressions.
- They cannot access variables other than those in their parameter list.