Escuela Politécnica Superior

UAM UNIVERSIDAD AUTONOMA DE MADRID

Introduction to Programming with Python

II.3. Functions, files, modules

Oscar Delgado
oscar.delgado@uam.es

Functions

Group of source code lines that can be executed by "calling" them from other point of the program

```
def print_hello(n):
    print('Hello ' * n)
    print()

print_hello(3)
print_hello(5)
times = 2
print_hello(times)
```

```
Hello Hello Hello
Hello Hello Hello Hello Hello
Hello Hello
```

Functions - Benefits

- Make large program easier to read and maintain
- All repeated code MUST be included in a function
- Useful to create your own utilities, math functions, etc.

Functions – Returning values

return statement can be used, with or without arguments, to return a value to the caller

```
def convert(t):
    return t*9/5+32

print(convert(20))
```

```
def multiple_print(string, n, bad_words):
    if string in bad_words:
        return
    print(string * n)
    print()
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Returning logical values

```
def is_even(n):
    if n%2==0:
      return True
    else:
      return False
```

```
def is_even(n):
    return n%2==0
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Returning multiple values

A function can return multiple values using a list

```
def solve(a,b,c,d,e,f):
    x = (d*e-b*f)/(a*d-b*c)
    y = (a*f-c*e)/(a*d-b*c)
    return [x,y]
xsol, ysol = solve(2,3,4,1,2,5)
print('The solution is x = ', xsol, 'and y = ', ysol)
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Managing logical values

How to check is a number is even?

```
if is_even(n) == True:
```

…

```
if is_even(n) == False:
```

…

```
if is_even(n):
```

…

```
if not is_even(n):
```

…

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Functions must be reusable

Write a function that print the last digit of a number

```
def last_digit(n):
    print(n%10)
```

```
def last_digit(n):
    return n%10
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Local and global variables

```
def func1():
    for i in range(10):
        print(i)

def func2():
    i=100
    func1()
    print(i)
```

Local and global variables

```
from math import sqrt

def triangle_area(a, b, c):
      s=(a + b + c)/2
      return sqrt(s*(s-a)*(s-b)*(s-c))
s = 4
print(triangle_area(s-1, s, s+1))
print(s)
print(a)
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Default arguments

A **default value** can be specified for an argument, that makes it optional (it the caller don't use it, it takes the default value)

```
def multiple_print(string, n=1)
    print(string * n)
    print()

multiple_print('Hello', 5)
multiple_print('Hello')
```

```
HelloHelloHelloHelloHello
Hello
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Keyword arguments

Order of arguments in functions with a lot of them can be difficult to remember:

```
def fancy_print(text, color, background, style, justify):
```

Python allows to name the arguments (and change their order) when calling a function:

```
fancy_print(text='Hi', color='yellow', background='black',
            style='bold', justify='left')
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Keyword arguments

Default and keyword arguments can be mixed (indeed it is recommended) :

```
def fancy_print(text, color='black', background='white',
                style='normal', justify='left'):
    # function code goes here

fancy_print('Hi', style='bold')
fancy_print('Hi', color='yellow', background='black')
fancy_print('Hi')
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Importing modules

```
from random import randint, choice
from random import *
import random
```

```
randint(1,10)
randint(1,10)
random.randint(1,10)
```

Program structure

```
<<modules import>>

<<functions definitions>>

<<main code>>
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Useful modules – Date and time

time module has some useful functions for dealing with time

sleep This function pauses the program for a specified amount of time (in seconds)

```
sleep(2)
sleep(.05)
```

Useful modules – Date and time

datetime module allows to work with dates and times together

```
from datetime import datetime

print(datetime.now())
```

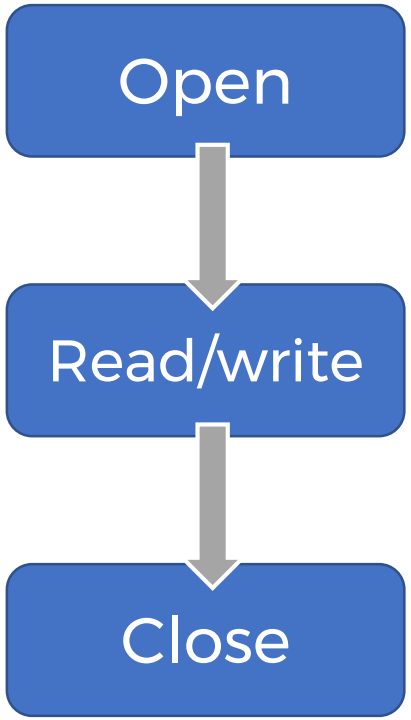
Literally, tens of functions to manage date&time. Look the official documentation

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Files & Directories

Files - Lifecycle




```
f = open("data.txt", "r")

for line in f:
    print(line)

f.close()
```

Never forget to close the file!!

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Read a whole file

If you file is small (hundreds or few thousands of lines), can be read in just one step:

```
f = open("data.txt", "r")
lines = f.readlines()
f.close()
```

readlines() reads all contents as a list of strings (for each line)

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Files - Writing

write() method writes any string to an open file, but it does not add a newline character ('\n')

```
f_output = open(“data.txt”, “w”)
f_output.write(“Hi there!\n”)
f_output.close()
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Filesystem management

Changing the directory A file is assumed to be in the same directory as your program itself. If not, you have to specify the directory, like below:

```
s = open('c:/users/heinold/desktop/file.txt').read()
```

If you have a lot of files that you need to read, all in the same directory, you can use os.chdir to change the directory. Here is an example:

```
os.chdir('c:/users/heinold/desktop/')
s = open('file.txt').read()
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Listing files

Getting the files in a directory The function **listdir()** returns a list of the entries in a directory, including all files and subdirectories

```
import os
directory = 'c:/users/heinold/desktop/'
files = os.listdir(directory)
for f in files:
    if os.path.isfile(directory+f):
        s = open(directory+f).read()
        if 'hello' in s:
            print(f)
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

More on files and directories

Function	Module	Description
mkdir(dir_path)	os	Create a directory
rmdir(dir_path)	os	Remove a directory
remove(file_path)	os	Delete a file
rename(old_name, new_name)	os	Rename a file from *old_name* to *new_name*
copy(src_file_path, dst_file_path)	shutil	Copy a file from *src_file_path* to *dst_file_path*

Running other programs

Running programs One way for your program to run another program is the **system()** function in the **os** module:

```
import os
os.chdir('c:/users/heinold/desktop')
os.system('file.exe')
```

Working with modules

A **module** is just a file containing Python definitions and statements

There are three main sources for modules:

- Some ‘standard’ modules (math, os) are available through the Python Standard Library
- Others can be installed with Python’s package manager, **pip**
- Creating and importing own modules

Installing new modules with **pip**

A de facto standard for Python's modules management

- Install a package

```
# pip3 install <<module_name>>
```

- List installed packages

```
# pip3 list
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Creating new modules

fibonacci.py

```
# return Fibonacci series up to n
def fib(n):
    result = []
    a = 0
    b = 1
    while b < n:
        result.append(b)
        c = a + b
        a = b
        b = c
    return result
```

my_program.py

```
import fibonacci as fibo


print(fibo.fib(20))
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID

Documenting code

```
# ———————————————————————
# Vectors module
# ———————————————————————
# Authors: Oscar Delgado
# ———————————————————————
# Provides constants and
functions for the vectorial
calculus in 3D dimensions.
# ———————————————————————
# Exported constants:
# vI, vJ, vK: unity vectors
```

```
# Exported functions
# vReadVector ():
# DESCRIPTION: read a vector from keyboard
# INPUT: no input parameters
# OUTPUT: returns a vector typed by user in
keyboard
#
# vShowVector (v):
# DESCRIPTION: prints the vector in the
screen with cartesian (x,y,z) notation
# INPUT: v -> vector to show
# OUTPUT: nothing to return
```

Escuela Politécnica Superior

UAM UNIVERSIDAD AUTÓNOMA DE MADRID