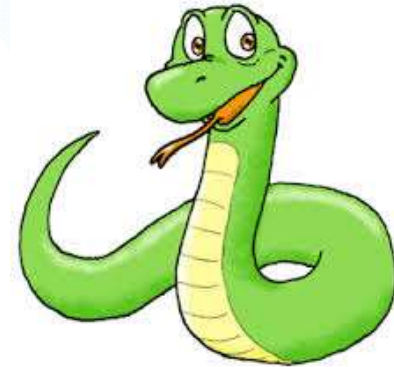




Python



1

VICKY PAPADOPOULOU LESTA, Assistant Professor, Member at AHPC group

MICHALIS KYPRIANOU, member of the AHPC group(internship project)

Department of Computer Science And Engineering
European University Cyprus

Workshop on Scientific Applications of Computing, 27th Oct. 2015

Python- General Information

2

- ▶ Designed from Guido van Rossum in 1991 at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- ▶ Derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, and Unix shell and other scripting languages.
- ▶ Python is copyrighted.
- ▶ Python source code is now available under the GNU General Public License (GPL).
- ▶ Latest version is 3.5.0

What is Python?

▶ Python is Interpreted

- ▶ Python is processed at runtime by the interpreter.

▶ Python is Interactive:

- ▶ You can interact with the interpreter directly to write your programs.

▶ Python is Object-Oriented:

- ▶ supports Object-Oriented style or technique of programming

▶ Python is a Beginner's Language:

- ▶ Python is a great language for the beginner-level programmers

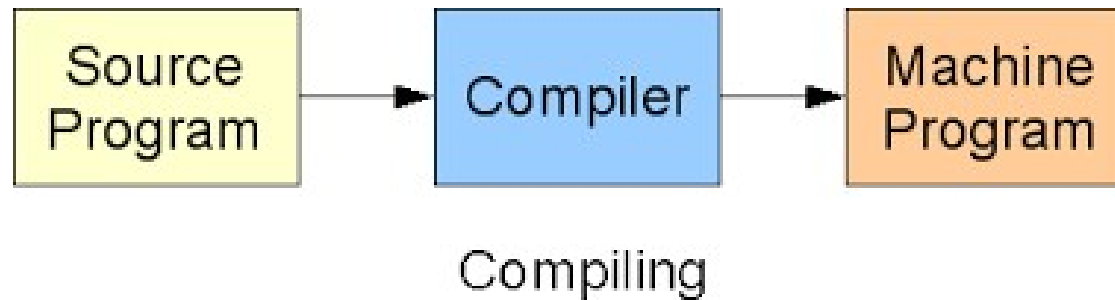
Applications for Python

4

- ▶ Web and Internet Development
- ▶ Scientific and Numeric
- ▶ Education
- ▶ GUIs
- ▶ System programming

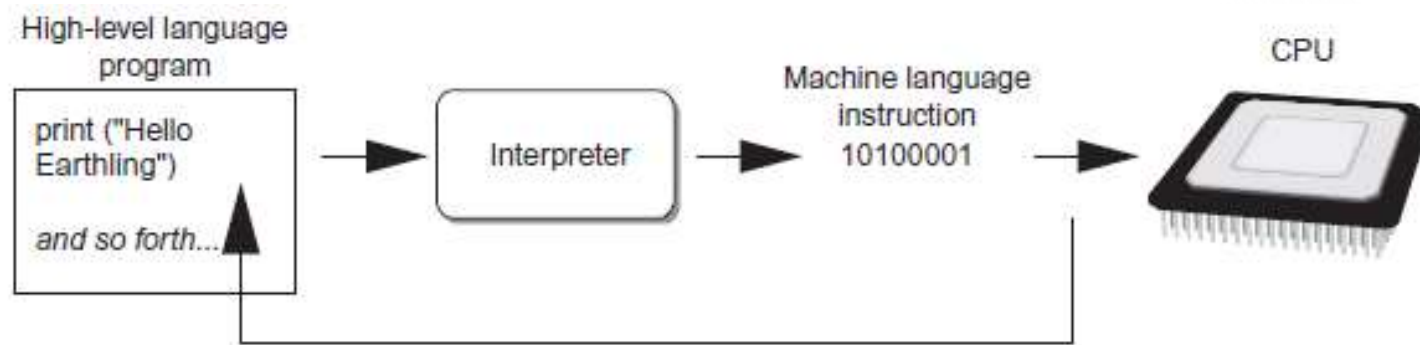
Compilers and Interpreters

- ▶ Programs written in high-level languages must be **translated into machine language** to be executed
- ▶ Compiler: translates a high-level language program into separate machine language program
 - ▶ Machine language program can be executed at any time



Compilers and Interpreters

- ▶ Interpreter: *translates* and *executes* instructions in high-level language program
 - ▶ Interprets one instruction at a time,
 - ▶ No separate machine language program



Python is an interpreted language

This process is repeated for each high-level instruction.

Why Python?

7

- ▶ Programs in Python are typically much shorter than equivalent C, C++, or Java programs, for several reasons:
 - ▶ The high-level data types allow you to express complex operations in a single statement;
 - ▶ Statement grouping is done by indentation instead of beginning and ending brackets;
 - ▶ No variable or argument declarations are necessary.

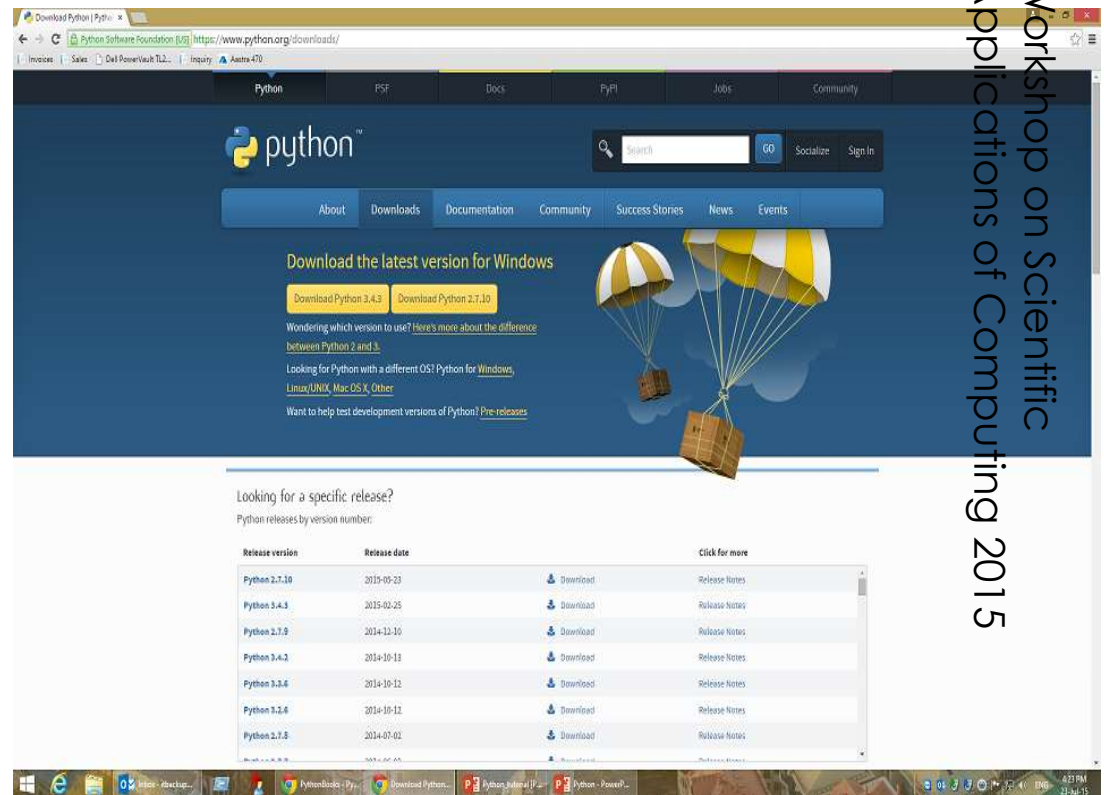
Get Python for free 😊

8

► It is an open source language

► Download for free:

<https://www.python.org/downloads/>

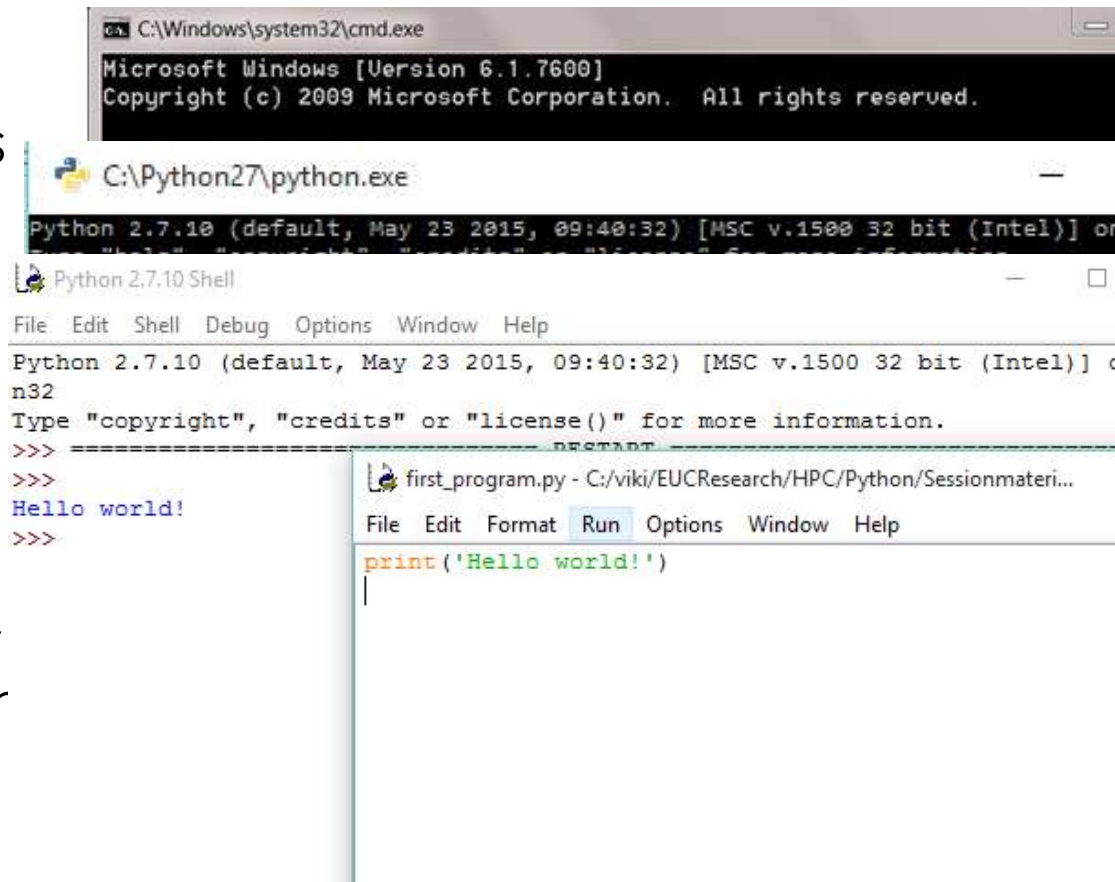


Workshop on Scientific Applications of Computing 2015

Workshop on Scientific Applications of Computing 2015

Python Interpreter:

- ▶ Read and Execute Python Programs
- ▶ Two modes:
 - ▶ **Script mode:**
 - ▶ Load and execute python programs
 - ▶ **Interactive mode**
 - ▶ Write statements and execute them in command line
- ▶ **IDLE** (Integrated **D**evelopment **E**nvironment) (Python GUI) provides both
 - ▶ Interactive mode
 - ▶ script mode



The screenshot shows two overlapping windows. The top window is a Windows command prompt titled 'C:\Windows\system32\cmd.exe' with the text: 'Microsoft Windows [Version 6.1.7600] Copyright (c) 2009 Microsoft Corporation. All rights reserved.' The bottom window is a Python 2.7.10 Shell titled 'C:\Python27\python.exe' and 'Python 2.7.10 Shell'. It has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The shell displays the Python version and date, followed by a prompt 'Type "copyright", "credits" or "license()" for more information.' Below this, there is a separator line and the text 'Python 2.7.10 Shell'. The user has entered 'Hello world!' and the shell has responded with 'Hello world!'. In the background, a portion of an IDLE editor window is visible, showing a file named 'first_program.py' with the code 'print('Hello world!')'.

Python Editors

10

- ▶ Some popular editor to write program in python are
 - ▶ Sublime Text
 - ▶ Vim
 - ▶ Emacs
 - ▶ Notepad++
 - ▶ TextWrangler

Anaconda Platform



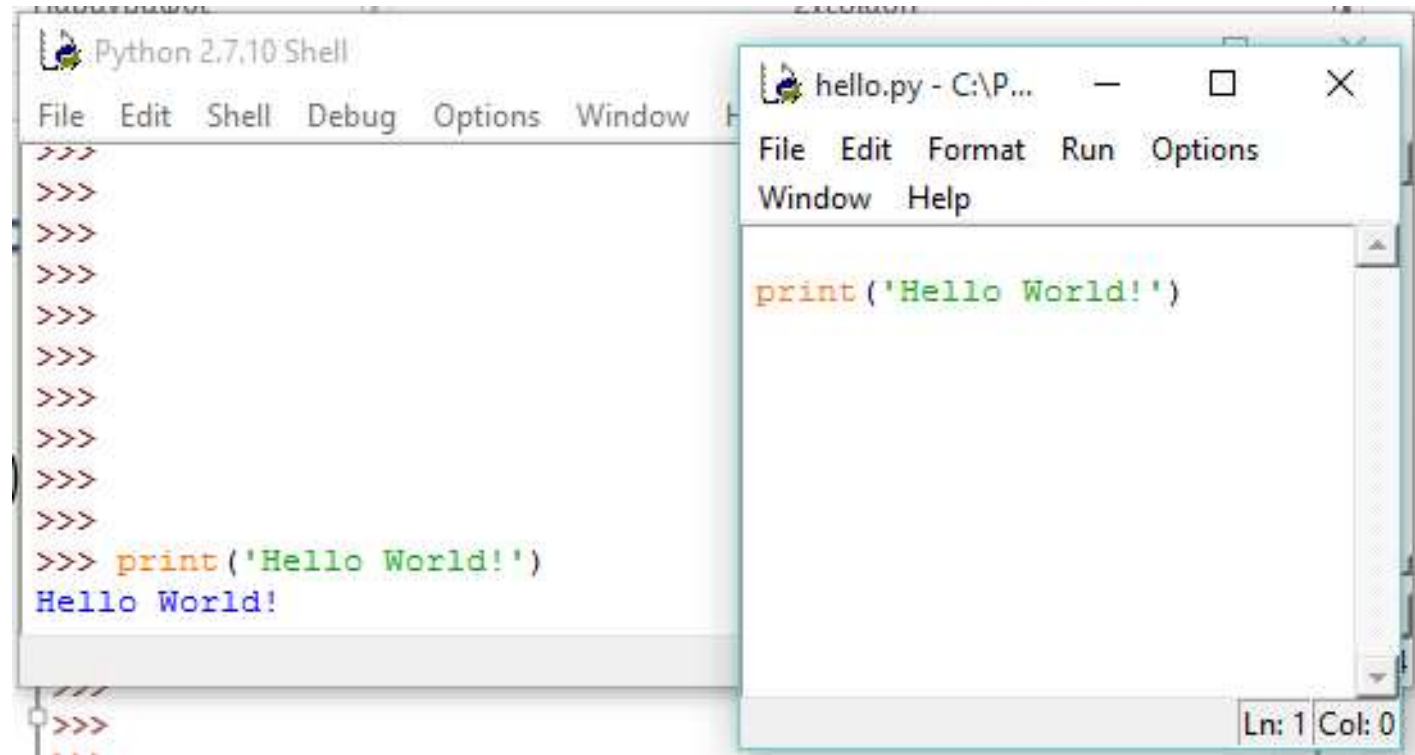
11

- ▶ FREE enterprise-ready Python distribution for data analytics, processing, and scientific computing.
- ▶ Anaconda comes with
 - ▶ Python 2.7 or Python 3.4 and
 - ▶ 100+ cross-platform tested and optimized Python packages.
- ▶ Get it free from:
 - ▶ <https://www.continuum.io/downloads>

Hello World in Python..

12

- ▶ print 'Hello World!'
- ▶ Help?!
 - ▶ help('statement')
 - ▶ Eg:
 - ▶ help('print')



The image shows two overlapping windows from a Python 2.7.10 environment. The background window is the 'Python 2.7.10 Shell' with a menu bar (File, Edit, Shell, Debug, Options, Window) and a command prompt interface. It shows several prompt lines '>>>' followed by the command `print('Hello World!')` which has been executed, resulting in the output 'Hello World!'. The foreground window is titled 'hello.py - C:\P...' and has a menu bar (File, Edit, Format, Run, Options, Window, Help). It contains a single line of Python code: `print('Hello World!')`. The status bar at the bottom right of the foreground window indicates 'Ln: 1 Col: 0'.

Simple Input-Output

13

► Example 1

```
n =input ('Give me a number..')  
print 'You gave me the number: ', n
```

► Example 2

```
n =input ('Give me a String..') # n = raw_input ('Give me a string..')  
print 'You gave me the string: ', n
```

Test it!

(inputOuputNumber.py)

- Give a number
- Give a string quoted
- Give a string *not* quoted

Variables and Input-Output

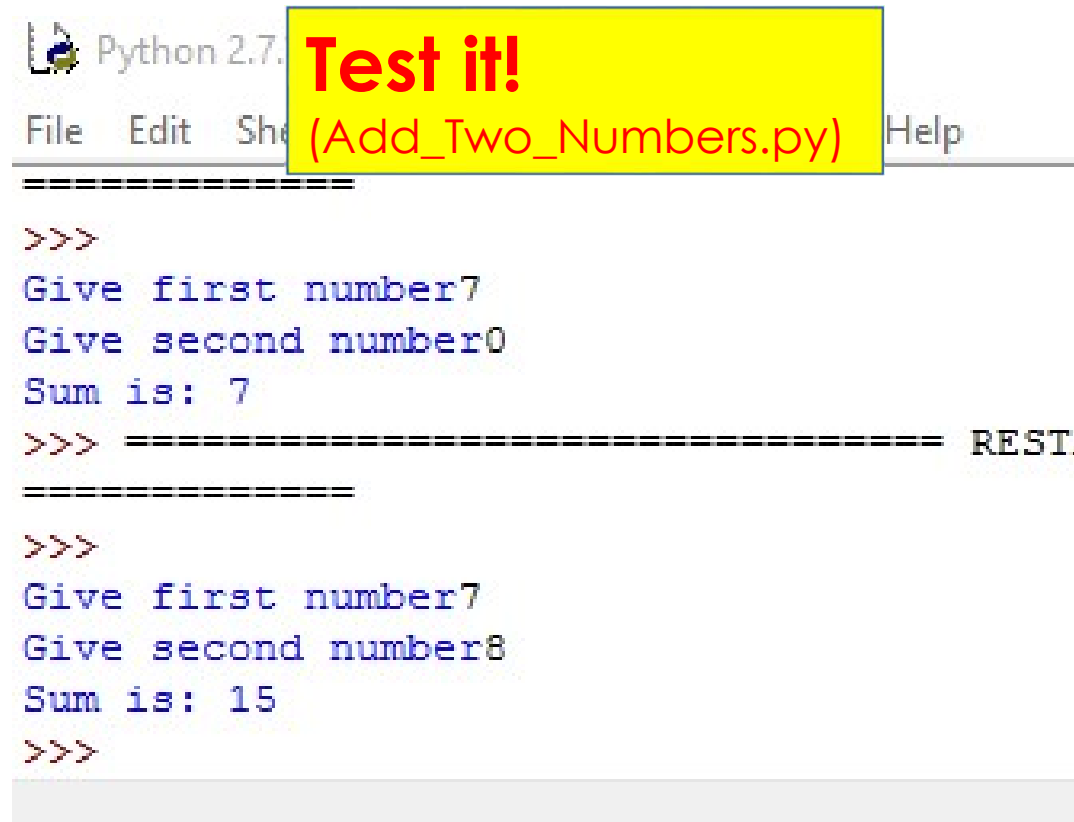
14

► Syntax:

► Variable = input ('message')

► Example

```
num1 = input('Give first number ')\nnum2 = input('Give second number ')\nsum = num1+num2\nprint 'Sum is:', sum
```



```
Python 2.7.6 Shell\nFile Edit Shell Help\n(Add_Two_Numbers.py)\n>>>\nGive first number7\nGive second number0\nSum is: 7\n>>> ===== REST\n>>>\nGive first number7\nGive second number8\nSum is: 15\n>>>
```

Variables

- ▶ A name that represents a value stored in the computer memory
- ▶ Declaration, creation of a variable is done by:
 - ▶ Assignment operation
 - ▶ Examples: `x=1` and `x="Python"`
- ▶ Garbage collection:
 - ▶ removal of values that are no longer referenced by variables
 - ▶ Carried out by Python interpreter
- ▶ A variable can refer to item of any type
 - ▶ Variable that has been assigned to one type can be reassigned to another type

Data Types

16

▶ **Data Type:**

- ▶ Strings
- ▶ Floats
- ▶ Integer
- ▶ Lists
- ▶ Tuples
- ▶ Sets
- ▶ Dictionaries

A note on Constant Variables..

17

- ▶ **There **no constant** variable in Python**
- ▶ **How can declare constant variable?**
 - ▶ Create a function
 - ▶ Declare local variables in a function
 - ▶ and use it as constant

Calculations, Arithmetic Operators

18

► **Examples:** You can write calculations in shell

► $10 + 5$ vs $(10.5 + 5.5)$

► 15 and 15.5

► $10 - 5$ vs $(10.5 - 5.5)$

► 5 vs 5.0

► $10 / 4$ vs $10.0 / 4$ (floating point division)

► 2 vs 2.5

► $9 // 4$ or $9.3 // 4$ (integer division)

► 2 and 2

► $10 \% 2$ vs $10.5 \% 2.5$ (remainder)

► 0 vs 0.5

► $10.5^{**}2$ (exponent operator)

► 110.25

Test it!

Mixed-Type Expressions and Data Type Conversion

- ▶ Data type resulting from math operation depends on data types of operands
 - ▶ Two `int` values: result is an `int`
 - ▶ Two `float` values: result is a `float`
 - ▶ `int` and `float`: `int` temporarily converted to `float`, result of the operation is a `float`

▶ Example 1

```
x=15.7
y=int(x)
print 'y = ',y
```

```
dataConversion.py - C:\Pyt
File Edit Format Run Op
x=15.7
y=int(x)
print('y = ',y)
```

Test it!

(Add_Two_Numbers.py dataConversion)

```
Python 2.7.10 (default, May 23 2015, 09:40:32) [MSC v.1
n32
Type "copyright", "credits" or "license()" for more inf
>>> ===== RESTART =====
>>>
y = 15
>>> |
```

Decision Statements

► Syntax:

► if <condition>:
 statements1

or :

elif <condition2>
 <statements3>

else:
 <statements2>

```
if (X!=10):
    print("Xis not eq
else
    print(("Xis equal
```

► IMPORTANT NOTE:

- The statement1/statements2 or *nested if stms* are specified by *indentation*.

Example2:

```
► if x==10
    print("X is equal to 10.)
elif x==9
    print("X is equal to nine.")
else:
    print("X is less or equal to eight. ")
```

Repetition structures: While Loop

21

► Syntax

► `while <expression>:`
 statements

► NOTE:

blocks are
identified through
indentation!



► Example

```
budget = float(input('Enter amount budgeted for the month: '))
while spent != 0:
    spent = float(input('Enter an amount spent(0 to quit)'))
    total += spent
if budget >= total:
    print 'You are $', (budget - total), 'under budget. WELL DONE!'
else budget < total:
    difference = total - budget
    print 'You are $', total - budget, 'over budget. PLAN BETTER NEXT TIME!'
```

Test it!
(whileLoopExample.py)

A count-control Repetition structure: For Loop

22

► Syntax

`for variable in [val1, val2, etc]:`
`statements`

➤ NOTE:

Nested blocks are
identified through
indentation!

► Example

```
totalRainfall=0
for month in [1,2,3,4,5,6,7,8,9,10,11,12]:
    monthRainfall = float(input('Enter the rainfall amount of \
the month:'))
    totalRainfall += totalRainfall
print'Total year's rainfall: ', totalRainfall, 'cms'
```

Test it!

(forLoopExample.py)

Using the range with the for Loop

► For syntax:

for variable in range
statements

► Example

```
totalRainfall=0
for month in range(1,13)
    print 'current month is ', month
    monthRainfall = float(input('Enter the rainfall amount of the\
month:'))
    totalRainfall += monthRainfall
print'Total year's rainfall: ', totalRainfall, 'cms'
```

► Range syntax:

- range(start, stop[, step])
 - Starting value is start
 - stop is not included
 - If the step is omitted, it defaults to 1.
 - If the start is omitted, it defaults to 0.

Test it!

(forLoopExample.py)

Control Statements

24

► Break Statement

- Stop Loop

► Example

```
► for letter in 'Python':  
    if letter == 'h':  
        break  
    print ("Current Letter :", letter)
```

► Output

```
► Current Letter :P  
Current Letter :y  
Current Letter :t
```


Control Statements

25

▶ Continue Statement

- ▶ Reject all the remaining statements and moves to the top of the loop

▶ Example

- ▶ for letter in 'Python':
 if letter == 'h':
 Continue
 print ("Current Letter :", letter)

▶ Output

- ▶ Current Letter :P
Current Letter :y
Current Letter :t
Current Letter :o
Current Letter :n

Modules and Packages

- ▶ A **module** is a file containing Python definitions and statements.
- ▶ Python comes with a standard library functions stored in modules
- ▶ **Example:** `random module`
- ▶ To import a module → `import a_module`
- ▶ **Packages:** a collection of modules
 - ▶ **Examples:** `Numpy, Scipy, Matplotlib`
- ▶ To import a package:
 - ▶ `import a_module` or `from a_module import something`
- ▶ **Example:**
 - ▶ `import numpy` or `from numpy import pi`

Using else Statement with Loops

27

- ▶ If the **else** statement is used with a **for/while** loop,
 - ▶ the **else** statement is executed when the loop has exhausted iterating the list/when the condition becomes false.

- ▶ While-else Syntax:

while <condition>:

statements

else:

statements

Hands-on Exercise 1!

Guessing Number game. (EX1_loops_Guess.py)

- Write a program where the user has to **guess** a number between a range of **1 to n** guessed by the program.
- The player inputs his guess.
- The program **informs** the player, if this number is **larger**, **smaller** or **equal** to the secret number.
- If the player wants to give up, he or she can input a **0** or a **negative** number.

Introduction to Functions

28

- ▶ **Function:** group of statements within a program that perform as specific task
 - ▶ Usually one task of a large program
 - ▶ Functions can be executed in order to perform overall program task

Void Functions and Value-Returning Functions

29

- ▶ **A void function:**

- ▶ Simply executes the statements it contains and then terminates.

- ▶ **A value-returning function:**

- ▶ Executes the statements it contains, and then it returns a value back to the statement that called it.
 - ▶ **Examples:** `input`, `int`, and `float` functions

Function Syntax

► Syntax:

► **Void Function:**

```
def fname (arguments)
    statements
```

► **Value-Returning Function:**

```
def fname (arguments)
    statements
    return <expression>
```

► **Call a function syntax:**

```
fname(value1,value2,etc)
```

NOTE:

► **Each block must be *indented***

- Lines in block must begin with the same number of spaces

Function Examples

31

1st Example

```
def test():  
    print("Hello Python")
```

```
test()
```

Output

Hello Python

2nd Example

```
def test(arg1,arg2):  
    return(arg1+arg2)
```

```
arg1 = 5
```

```
arg2 = 10
```

```
sum = test(arg1,arg2)
```

```
print ("sum is: ",sum)
```

Output

Sum is: 15

Examples – Call Functions

1st Way: by position

```
def test(arg1,arg2)  
    return(arg1+arg2)
```

```
arg1 = 5
```

```
arg2 = 10
```

```
sum = test(arg1,arg2)
```

```
print ("sum is: ",sum)
```

Output

Sum is: 15

2^d Way: by name

```
def test(arg1,arg2)  
    return(arg1-arg2)
```

```
sum = test(arg2=10,arg1=5)
```

```
print ("sum is: ",sum)
```

Output

Sum is: -5

Local Variables in Functions

33

- ▶ Local variable: variable that is assigned a value inside a function
 - ▶ Belongs to the function in which it was created
 - ▶ Only statements inside that function can access it

Passing Arguments to Functions

▶ **Argument:** piece of data that is sent into a function

▶ Function can use arguments in calculations

▶ Syntax - function definition:

```
def function_name(formal parameter):
```

▶ When calling the function, the argument is placed in parentheses following the function name

▶ Syntax- function call:

```
function_name(actual parameter):
```

Returning Multiple Values

- ▶ A function can return more than one values
- ▶ Syntax:
 - ▶ Return expression1, expression2, etc.
- ▶ When you call the function in an assignment statement, you need to use more than one variables:
 - ▶ first, second, etc = function_name (parameters)

Hands-on Exercise 2!

36

Test Grade and Average (EX2_grades.py)

- Write a program that asks the user to enter **five** test scores.
- The program should display a **letter grade** for each score and the **average** test score.
- Define two functions:
 - `Calc_average` function → to calculate the average of the scores
 - `Determine_grade` function → to calculate the average of the scores
- Note: letter score is a 10 point system (A = 90-100 B = 80-89, etc)

Python's Functions Passing Parameters Methods

- ▶ When a function is called, the parameters are passed using **call-by-value** or **call-by-reference**?
 - ▶ **Neither!**
- ▶ **Python uses "Call-by-Object", also called "Call by Object Reference" passing parameter method for the function calls.**
 - ▶ Passed objects of **mutable(changeable)** types can be changed by the called function
 - ▶ Passed objects of **immutable** types cannot be changed by the called function.
- ▶ Some **immutable** types:
 - ▶ int, float, long, complex, str, bytes, tuple
- ▶ Some **mutable(changeable)** types:
 - ▶ list, set, dict, byte array

Strings

► Syntax:

► **variable = "Txt"**

► Print:

► **print(variable[range])**

► Other Operations

- "hello"+"world" "helloworld" # concatenation
- "hello"*3 "hellohellohello" # repetition
- "hello"[0] "h" # indexing
- "hello"[-1] "o" # (from end)
- "hello"[1:4] "ell" # slicing
- len("hello") 5 # size
- "hello" < "jello" 1 # comparison
- "e" in "hello" 1 # search

Sequences

39

▶ Sequence:

- ▶ An object that contains multiple items of data
- ▶ The items are stored in sequence one after another

▶ Python provides different types of sequences, including

- ▶ *Lists*
- ▶ *Tuples*

▶ The difference between these:

- ▶ A list is **mutable** (changeable, variable) and a tuple is **immutable**
- ▶ **Syntax difference:** `[list]` vs `(tuple)`

Introduction to Lists

40

- ▶ **List:** an object that contains ordered data items
 - ▶ **syntax:** *list = [item1, item2, etc.]*
 - ▶ **Example:** `my_list = [10, 20, 30, 40]`
- ▶ **List element:** An item in a list
 - ▶ **syntax:** `list[item position]`
 - 🕒 Index of **1st** element in position **0** in the list
 - ▶ **Examples:** `my_list[1] → 20` , `my_list[4] → error`
- ▶ A list can hold items of different types
 - 🕒 E.g: `List=[1,2,'mike']`
- ▶ `print` function can be used to display an entire list

The Repetition Operator and Iterating over a List

► Repetition operator *****:

- makes multiple copies of a list and joins them together
- **Syntax: list * n**
- The symbol * is a repetition operator when applied to a sequence and an integer n
- **Example:** (a= 'Hello') a*2 → HelloHello

Test it!

► You can **iterate** over a list using a **for** loop

- **syntax: for x in list:**
- **Example:** a=[100,10,1,0]
 for n in a:
 print n

Test it!
(ListsIterating.py)

Lists Are Mutable

► **Mutable sequence:** the items in the sequence can be changed

- Lists are mutable, and so their elements can be changed

- **Example:**

```
numbers=[0]*5
```

```
index =0
```

```
While index < len(numbers):
```

```
    numbers[index]=99
```

```
    index+=1
```

Test it!

(ListsMutableExample.py)

List Slicing

43

- ▶ **Slice:** A span of items that are taken from a sequence
 - List slicing syntax: `list[start : end]`
 - Span is a list containing copies of elements from `start` up to, but **not including**, `end`
 - ▶ If `start` not specified, 0 is used for start index
 - ▶ If `end` not specified, `len(list)` is used for end index
 - Slicing expressions can include a **step value** and negative indexes relative to end of list

Example:

44

```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']  
letters[2:5] = ['C', 'D', 'E']  
letters[2:5] = []  
letters[:] = []
```

Test it!
(ListsSlicing.py)

Finding Items in Lists with the `in` Operator

- ▶ The **in** operator
 - ▶ determines whether an item is contained in a list
 - ▶ **Syntax:** *item in list*
- ▶ Returns **True** if the item is in the list, or **False** if it is not in the list
- ▶ Use the **not in** operator to determine **whether** an item **is not** in a list
- ▶ Example:

```
list = [1, 2, 3, 4]
```

```
3 in list
```

```
→ true
```

Test it!

List Methods and Useful Built-in Functions

▶ **append(*item*):**

- ▶ *used to add items to a list* – item is appended to the end of the existing list

▶ **index(*item*):**

- ▶ *used to determine where an item is located in a list*
 - Returns the index of the first element in the list containing item
 - Raises ValueError exception if *item* not in the list

List Methods and Useful Built-in Functions (cont'd.)

- ▶ **insert(*index*, *item*):**

- ▶ Used to insert *item* at position *index* in the list

- ▶ **sort():**

- ▶ Used to sort the elements of the list in ascending order

- ▶ **remove(*item*):**

- ▶ Removes the first occurrence of *item* in the list

- ▶ **reverse():**

- ▶ Reverses the order of the elements in the list

Two or more Dimensional Lists

48

- ▶ **Two-dimensional list:** A list that contains other lists as its elements
 - 🍪 Also known as nested list

Example

▶ matrix = $[[1,2,3,4],$
 $[5,6,7,8],$
 $[9,10,11,12]]$

Vector structures of Numpy Package

49

- ▶ Use: `from numpy import array`
- ▶ Example
 - ▶ `r=array([1,2,3])`
 - ▶ `u =array([-1,-2,-3])`
 - ▶ `r+u` → addition of two arrays
 - ▶ `r*u` → dot product of two arrays!
 - ▶ `a-10`
 - ▶ `(a+3)*2`
- ▶ Use: `from numpy import array`
- ▶ Example
 - ▶ `r=matrix([1,2,3])`
 - ▶ `u =matrix ([-1,-2,-3])`
 - ▶ `r+u` → addition of two arrays
 - ▶ `w=r*u.T` → matrix multiplication!

Tuples

50

- ▶ **Tuple:** An **immutable** sequence
 - ▶ Very similar to a list
 - ▶ Once it is created it **cannot be changed**
- ▶ **Syntax:** `tuple_name = (item1, item2)`
 - ▶ `:` `()` instead of `[]` to distinguish from lists
- ▶ **Tuples support operations as lists**
 - ▶ Subscript indexing for retrieving elements
 - ▶ Methods such as `index`
 - ▶ Built in functions such as `len`, `min`, `max`
 - ▶ Slicing expressions
 - ▶ The `in`, `+`, and `*` operators

Tuples (cont'd.)

▶ Tuples do not support the methods:

- ▶ append
- ▶ remove
- ▶ insert
- ▶ reverse
- ▶ Sort

▶ Example

- ▶ `tup1 = (12, 34.56);`
- ▶ `tup2 = ('abc', 'xyz');`
- ▶ `print 'tup1 = ', tup1[0:1]`

Test it!
(ListsSlicing.py)

Dictionaries

- ▶ **Dictionaries:** *object that stores a collection of data*
 - ▶ Each element consists of a **key** and a **value**
 - ▶ Often referred to as **mapping of key to value**
 - ▶ **Key must** be an **immutable** object
 - ▶ Dictionary is **mutable**
- ▶ To retrieve a specific value, use the key associated with it
- ▶ **Syntax:** `dictionary = [key1:value1, key2:value2...]`
 - ▶ `dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};`
- ▶ **Example:**

```
print "dict['Name']: ", dict['Name']  
print "dict['Age']: ", dict['Age']
```

Test it!

Sets

- ▶ **Sets:** object that stores a collection of data in same way as mathematical set
 - ▶ All items must be **unique**
 - ▶ Set is **unordered**
 - ▶ Elements can be of different data types
 - ▶ There are **mutable**
- ▶ Sets support mathematical operations (union, intersection, difference)
- ▶ **Syntax:** `set = {item1,item2,...}`
- ▶ **Example:**

```
a = set('abracadabra')  
b = set('alacazam')  
a-b
```

Test it!

Built-in Functions

54

- ▶ The Python interpreter has a number of functions and types built into it that are always available.
- ▶ You can use a module of functions using **import**:
- ▶ Eg.
 - ▶ `import math`

Built-in Function

► Table with built- in Function

Built-in Functions				
<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

Reading and Writing Files

56

► Syntax

► `f = open(filename,mode)`

filename = ' name of file that we want to have access

mode = 'optional string that specifies the mode in which the file is opened'

► Example

► `f=open('text.txt','r')`

Reading and Writing Files-Continue

57

► Mode

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)
'U'	Universal newlines mode (deprecated)

More on Modules

- ▶ Split your program in a several files, called **module**.
- ▶ Module can be **imported** into other modules or into the main module

▶ Syntax

- ▶ `import filename.py`

▶ To install a module:

- ▶ `pip install module_name`

- ▶ **Example:** `pip install numpy`

▶ Create your own module: (example1.py)

- ▶ **Example:**

```
def name(n)
    print (n)
def surname(s)
    print (s)
```

Classes - Object

59

▶ **Class**

- ▶ Is an extensible program-code-template for creating object

▶ **Object**

- ▶ Is a location in memory having a value and possibly referenced by an identifier.
- ▶ Can be variable, data structure, function

Classes

60

► Syntax

► `class class_name:`
 statements

► Example

```
class Coin:
    def __init__(self):
        self.sideup = 'Heads'
    def toss(self):
        if random.randint(0, 1) == 0:
            self.sideup = 'Heads'
        else:
            self.sideup = 'Tails'
    def get_sideup(self):
        return self.sideup
```

Test it!

Classes_coin.py

Classes

61

► Initializer method-Example:

► `def __init__(self, var1, var2)`

Self parameter:
references the specific
attribute

`self.v1=var1`
`self.v2=var2`

Syntax:

`def __init__(self, vars):`

► Example

```
► class c_name:
    def __init__(self, num1, num2)
        self.n1=num1
        self.n2=num2

    def sum...
```

- Attribute variables are **public**
- Can be accessed only with: `className.variable_name`

Classes (cont.)

- ▶ To create a new instance of a class call the initializer method
 - ▶ **Format:** `My_instance = Class_Name()`
 - ▶ **Example:** `my_coin=Coin()`
- ▶ To call any of the class methods using dot notation:
 - ▶ **Format:** `My_instance.method()`
 - ▶ **Example:** `my_coin.toss()`
- ▶ To make an attribute private
 - place two underscores (__) in front of attribute name
 - ▶ **Example:** `self.sideup = 'Heads'`

Test it!

Hands-on Exercise 3!

63

Test Grade and Average

- Write a class named Car that has the following data attributes:
 - `__year_model` (for the car's year model), `__make` (for the make of the car) and `__speed` (for the car's current speed).
- The Car Class should have an `__init__` method that accepts the car's year model and make data attributes. It should also assign 0 to the `__speed` data attribute.
- The class should also have the following methods:
 - `accelerate` - the `accelerate` method should add 5 to the speed
 - `brake` - the `brake` method should subtract 5 from the speed
 - `get speed` - the `get_speed` method should return the current speed
- *Next, design a program that creates a Car object, and then calls the `accelerate` method five times. After each call to the `accelerate` method, get the it. Then call the `brake` method five times. After each call to the `brake` method, get the current speed of the car and display it.*