PYTHON PROGRAMMING

INTRODUCTION

- Iulia Chiriac
- Contacts

- iulia.chiriac.a@gmail.com
- http://luxoft-training.ru
- http://luxoft-training.com

TRAINING ROADMAP: STRUCTURE

- 40 hours
- 10-15 mins breaks every 1.5 – 2 hours
- Lunch break
- In-class individual practice
- In-class group workshops

SECTION 1: INTRODUCTION

WHY PYTHON?

- Very popular, top 10 programming languages

  over 1 million questions asked on stackoverflow.com;

  #1 by CodeEval, #3 by IEEE Spectrum
- Concise, clear, highly readable
- Dynamic, high level, interactive language
- Easy to embed as scripting
- Portable: Windows, Linux, Mac, Android, browser

EXECUTION MODEL VARIATION

- CPython – standard implementation
- PyPy – alternative interpreter and JIT compiler (EU funding)
- Jython – written in Java
- IronPython – written in C# and integrated with .NET
- Cython – Python + C = ♥

RUNNING PYTHON SCRIPTS: INTERACTIVE CONSOLE

- IDLE (local, usually installed by default)
- iPython (local)
- DreamPie (local)
- repl.it/languages/Python (online, inside a sandbox)
- live.sympy.org (online, inside a sandbox)
- shell.appspot.com (online, inside a sandbox)
- pythonanywhere.com (persistent sessions!)

EXECUTING PYTHON CODE

- Run a Python script as a file
    - Under Windows with CMD prompt
    - Under Linux/Unix/MAC
- Run a Python script from the interactive mode

```
Windows:
C:\Python27\python.exe C:\Scripts\script1.py

Linux/Unix:
chmod +x scripts/script1.py
python  scripts/script1.py
```

```
Windows:
C:\Users>python.exe
Python 2.7.10 (default, May 23 2015, 09:40:32) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>

Linux:
user@hostname:~ python
Python 2.7.2+ (default, Oct  4 2011, 20:03:08) [GCC 4.6.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

BASIC SYNTAX

- Comments

```
# this is a comment
```

- Literal constants
- Variables
- Logical vs Physical line
- Indentation
- built-in functions:

```
help() dir() type() print()
```

BASIC TYPES: NUMERIC

- Integers – int()

  Unlimited precision (in Python3; in Python2 at least 32 bits precision)

- Float – float()

  Precision : sys.float_info

BASIC TYPES: BOOL

- `bool()`
- subtype of int
- constants: True and False

```
>>> bool(1)
 True
 >>> bool(0)
 False
```

NUMERIC TYPES: OPERATIONS

- 0b... are numbers in base 2 ; 0o… are octal ; 0x... are numbers in hex
- Built-in operations: sum, pow, round, min, max
- Math operations: trunc, floor, ceil, exp, log, sqrt, factorial, fsum, sin, cos
- Math constants: math.pi, math.e
- Conversion: int(x, base), long(x, base), float(x), complex(real, imag)
- Convert into base: bin(nr), oct(nr), hex(nr)
- Not all python objects can be converted into numbers

NUMERIC TYPES:PYTHON3 CHANGES ON NUMBER ARITHMETIC

- Division operator

  Python 2

  - 3 / 2 => 1
  - 3 // 2 => 1
  - 3 / 2.0 => 1.5

  Python 3

  - 3 / 2 => 1.5
  - 3 // 2 => 1

BASIC TYPES: STRINGS

- List of characters, delimited by quotation marks: "" or ''
- Strings are immutable
- Multiline strings : """ """ or ''' '''
- Special characters should be escaped: ' , " , \
- \n – newline ; \t - tab

STRINGS: RAW STRINGS

- A string that ignores all escape characters and prints any backslash.
- Syntax:

```
s = r'Raw string - \'will be printed'
```

STRINGS: BYTESTRING VS STRING IN PYTHON3

- String

  Sequence of characters, human readable

  To write it on the disk it has to be converted to a bytestring

- Bytestring

  Sequence of bytes, non-human readable

  Similar to ‘unicode’ from Python2

- Methods: encode() , decode()

STRING TYPE: STRING MANIPULATIONS FUNCTIONS

- Accessing characters: [index]
- Slicing: [index : index] (count starts from 0, -1 last element)
- Concatenate: string1 + string2 (use += for appending)
- Multiply: string * number
- Length: len(string)
- ‘in’ and ‘not in’ operators

STRING TYPE: STRING MANIPULATIONS METHODS

- Finding: find(word[,start[,end]]), startswith(prefix[,start[,end]]), endswith(sufix[,start[,end]])
- Removing space: strip([chars]), lstrip([chars]), rstrip([chars])
- Joining: join(iterable)
- Splitting: split([separator[,maxsplit]])
- Changing letters: replace(old,new[,count]), upper(), lower(), title(), capitalize()
- Verifying the string nature: isupper(), islower(), isalpha(), isalnum(), isspace(), isdigit()
- Counting: count()
- Justifying text: rjust(width[,fillchar]), ljust(width[,fillchar]), center(width[,fillchar])

STRING TYPE: STRING MANIPULATIONS

```
#Single/double quotes example:
str1 = 'String 1'
str2 = "String's reloaded"
str3 = "\"Yes\", he said."

#'Raw' string example:
rstr = r"C:\Program Files"; print(rstr)
nrstr = "C:\Program Files"; print(nrstr)

#Concatenated strings:
word = 'Hello ' + 'world';
print(word)
print(word*2) # Prints string two times
```

```
#Slice operator:
str1 = 'Hello World!'
print(str1[0]) # Prints first character of the string
print(str1[2:5]) # Prints characters starting from 3rd to 5th
print(str1[2:]) # Prints string starting from 3rd character


#String methods
str2 = " Hello World! "
print(str2.strip()) # Remove leading and trailing spaces
print(str2.upper()) # Letters are converted to upper case
print(str2.split()) # Split string by spaces
print(str2.replace("World","Europe")) # Replace characters
print(str2.count("l")) # Occurrences of substring 'l' in string
```

BASIC TYPES EXERCISES

1. Given an integer number, print its last digit.
2. Given a three-digit number, find the sum of its digits.
3. Given the integer N - the number of minutes that is passed since midnight - how many hours and minutes are displayed on the 24h digital clock? The program should print two numbers: the number of hours (between 0 and 23) and the number of minutes (between 0 and 59).

   For example, if N = 150, then 150 minutes have passed since midnight - i.e. now is 2:30 am. So the program should print 2 30.

BASIC TYPES EXERCISES

4. Given the string s = "bandana":

- check if string "and" is contained in s
- find the index of the following strings: "n", "q"
- how many times does the string "an" appear in s?
- check if s is alphanumeric
- transform s to all uppercase
- check other string methods and try them out

BASIC TYPES EXERCISES

5. Given a string, print the following:

- In the first line, print the third character of this string.
- In the second line, print the second to last character of this string.
- In the third line, print the first five characters of this string.
- In the fourth line, print all but the last two characters of this string.
- In the fifth line, print all the characters of this string with even indices (remember indexing starts at 0, so the characters are displayed starting with the first).
- In the sixth line, print all the characters of this string with odd indices (i.e. starting with the second character in the string).
- In the seventh line, print all the characters of the string in reverse order.
- In the eighth line, print every second character of the string in reverse order, starting from the last one.

CONTROL FLOW

- if/elif/else
- while/else
- for

BASIC CONTROL STRUCTURES: IF/ELIF/ELSE

- Syntax

```
if (condition1):
    statement1
elif (condition2):
    statement2
elif (condition3):
   statement3
else:
   statement4
```

- else and elif are optional
- Single line if : *if (condition1):* statement 1

BASIC CONTROL STRUCTURES: WHILE/ELSE

- Syntax

```
while (conditions):
    statement1
    statement2
else:
    statement3
```

- Else is optional and will be executed when condition becomes false
- Single line while: *while (conditions):* statement 1

BASIC CONTROL STRUCTURES: FOR LOOP

- Syntax

```
for iter_variable in sequence:
    statement1
    statement2
else:
    statement3
```

- Else is optional and will be executed when iteration is completed

BASIC CONTROL STRUCTURES: LOOP CONTROL STATEMENTS

- break

  The loop terminates and the execution is transferred to the statement that follows the loop.

- continue

  The remaining statements will be skipped and the condition will be retested prior to reiterating.

- pass

  when a statement is required syntactically but you do not want any command or code to execute.

FUNCTION SYNTAX

- Syntax:

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

- return is optional

FUNCTION SYNTAX: ARGUMENTS

- Required arguments
  - Arguments passed to a function in correct positional order
  - The number of arguments in the function call should match exactly with the function definition
- Keyword arguments
  - They are related to the function calls
  - In a function call, the caller identifies the arguments by the parameter name
- Default arguments
  - An argument that assumes a default value if a value is not provided in the function call
- Variable-length arguments
  - They are used when you need to process a function for more arguments than you specified while defining the function.
  - These are not named in the function definition, unlike required and default arguments.

BUILT-IN FUNCTIONS

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

Information taken from docs.python.org

SIMPLE SCRIPTS

1. Write a function that takes a number and prints its square.
2. Write another function that takes a number as a parameter and returns the square. Are the results of the two functions different?
3. Write a Python program which iterates the integers from 1 to 50. For multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".
4. Write a function that prints the odd numbers from a given interval. The default interval is 0-100.

SIMPLE SCRIPTS

5. Write a function that builds html tags. Apply html escaping for html special chars.

   The function will receive 2 parameters – tag type and tag content. It will return the generated html as text.

   eg: f('b', 'Ham&Eggs') returns "<b>Ham&amp;Eggs</b>"

   HTML char escaping:

   - < becomes &lt;
   - > becomes &gt;
   - " becomes &quot;
   - & becomes &amp;

6. Write a program to read a string and to convert it to a bytestring encoded by utf-8.

SECTION 2: ITERABLES

SEQUENCE TYPES: LIST, TUPLE

- List:
  - Members might have different types of data
  - Creating:

```
l = list()   # empty list
l = []   # empty list
l = list(iterable)   # new list initialized from iterable's items
l = [element1, element2]   # list with two elements
```

  - Mutable
- Tuple
  - Members might have different types of data
  - Creating:

```
tuple(), (), (element1, element2), tuple(iterable)
```

  - Immutable

SEQUENCE TYPES: SLICE OPERATOR

- Slice operation returns a new list containing the requested elements
- Operations:
  - Copy of the list: lst[:]
  - Returns slices of the list
  - Modifies the size and elements of the list

SEQUENCE TYPES: SLICE OPERATOR

```
l1 = ['abcd', 786, 2.23, 'john', 70.2]

print l1[0] # Prints first element of the list -> abcd
print l1[1:3] # Prints elements starting from 2nd till 3rd -> [786, 2.23]
print l1[2:] # Prints elements starting from 3rd element -> [2.23, 'john', 70.2]
print l1[::-1] # Reverses the list

#Example of slice assignment:
l1[0:2] = [1, 2] # Replace some items
print(l1) # Prints [1, 2, 2.23, 'john', 70.2]

l1[0:2] = [] # Remove some items
print(l1) # Prints [2.23, 'john', 70.2]

l1[1:1] = ['insert1', 'insert2'] # Insert some items
print(l1) # Prints [2.23, 'insert1', 'insert2', 'john', 70.2]

l1[:0] = l1 # Insert a copy of itself at the beginning
print(l1) # Prints [2.23, 'insert1', 'insert2', 'john', 70.2, 2.23, 'insert1', 'insert2', 'john', 70.2]

l1[:] = [] # Clear the list
print(l1) # Prints []
```

LIST/SEQUENCE FUNCTIONS: MANIPULATION ELEMENTS

- list.append(*x*)

  Add an item to the end of the list; equivalent to a[len(a):] = [x]
- list.extend(*L*)

  Extend the list by appending all the items in the given list; equivalent to a[len(a):] = L
- list.insert(*i*, *x*)

  Insert an item at a given position. (a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x))
- list.remove(*x*)

  Remove the first item from the list whose value is *x*. It is an error if there is no such item.
- list.pop([*i*])

  Remove the item at the given position in the list, and return it.

  If no index is specified, a.pop() removes and returns the last item in the list.

LIST/SEQUENCE FUNCTIONS: SORTING AND COUNTING

- list.sort()

  Sort the items of the list, in place.
- list.reverse()

  Reverse the elements of the list, in place
- list.index(*x*)

  Return the index in the list of the first item whose value is *x*. It is an error if there is no such item.
- list.count(*x*)

  Return the number of times *x* appears in the list.

LISTS EXERCISES

1. Write a Python program to convert a list of characters into a string.
2. Given a list of numbers with all of its elements sorted in ascending order, determine and print the quantity of distinct elements in it.
3. Write a function that receives two lists as parameters and returns `True` if they have at least one common element, `False` otherwise.
4. Write a function `filter_long_words()` that receives as parameters a list of words and an integer `n` and returns the list of words longer than `n`.

SEQUENCE TYPES: SET, FROZENSET

- Unordered collection of hashable objects
- Used for testing memberships, removing duplicates, math operations
- Set type is mutable
- Frozenset type is immutable

SEQUENCE TYPES: SET, FROZENSET

- len(s) : number of elements
- x in s   : verifies membership of x in s
- x not in s: verifies that x in not a member of s
- issubset() or s1 <= s2 or s1 < s2
- issuperset() or s1 > other or s1 >= other
- isdisjoint()
- union() or s1 | s2
- intersection() or s1 & s2
- difference() or s1 – s2
- copy()

SEQUENCE TYPES: SET, FROZENSET

- add()
- remove()
- discard()
- pop()
- clear()
- update() or s1 |= s2
- intersection_update() or s1 &= s2
- difference_update() or s1 -= s2

SET EXERCISES

1. Given two lists of numbers. Count how many unique numbers occur in both of them.
2. Write a function that receives a text (a string) and returns the number of unique words. A word=a sequence of characters that is not whitespace (space, newline, tab).

MAPPING TYPES: DICTIONARY

- An unordered set of *key : value* pairs
- Creating:

Empty dict:	d = dict(); d = {}
Key by key:	d = dict(); d[key] = value
Compact :	d = { key:value, key1:value1}
Using keyword arguments:	d = dict(key=value,key1=value1)

- Deleting:

A key:	del d[key]

THE DICTIONARY TYPE: RESTRICTIONS

- Key has to be hashable
  - Number
  - String
  - Tuple that contains only numbers, strings or tuples
- Key is unique within the dictionary
  - If storing an existing key, the old value will be overwritten

THE DICTIONARY TYPE: GETTING KEYS/VALUES/ITEMS

- key in d: check whether *d* has a key *key*
- dict.get(key, default=None): For *key* key, returns value or default if key not in dictionary
- dict.items() :Returns a list/view of *dict*'s (key, value) tuple pairs
- dict.keys() : Returns list/view of dictionary dict's keys
- dict.values(): Returns list/view of dictionary *dict2*'s values

THE DICTIONARY TYPE: CLEAR/COPY/UPDATE

- dict.clear(): Removes all elements of dictionary *dict*
- dict.copy(): Returns a shallow copy of dictionary *dict*
- dict.pop(key[, default]): If *key* is in the dictionary, remove it and return its value, else return *default*
- dict.update(dict2): Adds dictionary *dict2*'s key-values pairs to *dict*

THE DICTIONARY TYPE: DICTIONARY ITERATORS

- Loop the keys of a dictionary:

```
for key in dict:
for key  in dict.keys():
```

- Loop keys and values of a dictionary:

```
for key, value in dict.items():
```

DICT EXERCISES

1. Given the following dictionary:

```
d = {'times': 100,
     'name': 'George',
     'hobbies': ['fishing', 'hiking']}
```

   - add key *'friends'* to *d* with *['Andrei', 'Mihai', 'Alina']* as value
   - sort value for key '*friends'*
   - remove *'hiking'* from hobbies list
   - remove *'times'* key from *d*

2. Given a list of strings build a dictionary that has each unique string as a key and the number of appearances as a value.
3. Write a python program to map two lists into a dictionary.

FUNCTIONS: VARIABLE LENGTH ARGUMENTS

```
def func(*args, **kwargs):
        for i in args:
                print(i)
        for k, v in kwargs.items():
                print(k, v)
```

- *args: variable-length positional arguments
- **kwargs: variable-length keyword arguments
- can be used when calling a function as well

STRINGS: PRINTING FORMATTED OUTPUT

- String objects have a specific method used for formatting – `format()`
- Format method uses formatters atoms to decide how to display data types along static strings
- Example:

```
"The result is: {formatter}".format(result_value)
```

STRINGS: PRINTING FORMATTED OUTPUT: FORMATTER

- Formatter syntax:

{[value]![conversion_method]:[fill][align][sign][width].[precision][format_type]}

- [value]: <empty>, Named, Indexed
- [conversion_method]: s – str(); r – repr()
- [align]: “<“ , “>“ , “=” , “^”
- [sign]: + , - ,‘ ’ [space]
- [format_type]:

    for strings(‘s’,None);

    for integers(‘b’,’c’,’d’,’o’,’x’,’X’,’n’,None);

    for float(‘e’,’E’,’f’,’F’,’g’,’G’,’n’,’%’)

STRINGS: PRINTING FORMATTED OUTPUT: FORMATTER

```
print 'Formatting with positional and named arguments'
print '{} {}'.format('one', 'two')
print '{1} {0}'.format('one', 'two')
print '{} {}'.format(1, 2)
print '{first_name} {last_name}'.format(last_name='Picard', first_name='Jean-Luc‘)

print 'Add some padding'
print '{:_<10}'.format('test')
print '{:_>10}'.format('test')
print '{:_^10}'.format('test')

print 'Add specific formatting to named value'
print '{result!s:_^20}'.format(result=(1, 2))
print '{result!r:_^20}'.format(result=(1, 2))

print '{result:_^ 20.2f}'.format(result=3.141)
print '{result:_^ 20.2f}'.format(result=-3.141)
```

SECTION 3:
OBJECT ORIENTED PROGRAMMING

OOP BASICS: CLASSES AND INSTANCES

- Class: template for creating objects
- Instance: object created according to a *template*

MEMBER ATTRIBUTES

- Instance attributes : owned by an instance of a class
- Class attributes: shared by all instances of a class

MEMBER FUNCTIONS

- Constructors
  - __init__()
- Destructors
  - __del__()
- Writing member functions
  - Self argument

MEMBER ATTRIBUTES – ACCESS CONTROL SOLUTIONS

- Public variables: attr
- Private variables: _attr

    By convention, they are treated as non-public parts of the API

    Should be considered an implementation detail and subject to change without notice.

- Private variables: __attr

    Accessible only within the class

    Name mangling for accessing outside the class: obj._class__attr

MEMBER ATTRIBUTES – GETTER/SETTER METHODS

- Getter: method that gets an attribute value
- Setter: method that sets an attribute value
- Deleter: method that deletes an attribute value
- Also, used by property class or as decorators

  @attr.setter, @attr.getter, @attr.deleter

MEMBER ATTRIBUTES – THE PROPERTY CLASS

- Returns a property attribute
- Syntax:

  class property([fget[, fset[, fdel[, doc]]]])

  - fget: getting an attribute
  - fset: setting an attribute
  - fdel: deleting an attribute
  - doc: docstring for the attribute

  @property

CLASS/FUNCTION DECORATORS

- @staticmethod:

  neither the object, nor class is passed as the first argument

  normal functions, called from an instance or from the class

- @classmethod:

  the class is the first argument of the method

OOP EXERCISES

1. Create a BankAccount class that receives two parameters on initialisation:
   - bank name (str)
   - amount of money (int)
2. Create two methods in this class, one to withdraw money and another one to deposit money into the account. The withdraw method will not allow withdrawing more money than available and it will print an error message when you attempt to do that.
3. Create a class Employee with three instance attributes:
   - person name (str)
   - bank account (BankAccount)
   - salary (default 0) (int)

Salary should be private. Create a property to set salary; the getter should return None.

Create a method receive_salary that will deposit in the employee's bank account an amount equal to its salary.

PYTHON CLASS TEMPLATE: INHERITANCE

- Syntax:

  class SubClassName (ParentClass1[, ParentClass2, ...]):

- Calling parent methods

  *super().method_name(*args, **kwargs)*

- Multiple class inheritance: C3 Method resolution order

  cls.__mro__

PYTHON CLASS TEMPLATE: INHERITANCE

```
class ItemList(object):
    def __init__(self, name):
        self.name = name
        self.description = None
        self.items = list()

    def add_item(self, item):
        self.items.append(item)

    def set_items(self, items):
        self.items = items

    def get_items(self):
        return self.items
```

```
class TodoList(ItemList):
    def __init__(self, name, asignee):
        # Call Super constructor(name) -- Python2.x
        # super(TodoList, self).__init__(name)
        # Call Super constructor(name) -- Python3.x
        super().__init__(name)
        self.asignee = asignee
    def __str__(self):
          return "Override 'str' in TodoList class"

it1 = ItemList("Item1")
it1.add_item("t1")
print(it1.get_items())
td1 = TodoList("ToDo1","User1")
print(td1.get_items())
td1.add_item('todo1') ; print(td1.get_items())
print(str(td1)); print(str(it1))
```

OPERATORS – RELATED FUNCTIONS

- comparison: __eq__, __ne__, __lt__, __gt__, __le__, __ge__
- unary operators: __pos__, __neg__, __abs__, __invert__, __round__, __floor__, __ceil__, __trunc__
- arithmetic operators: __add__, __sub__, __mul__, __floordiv__, __div__, __truediv__, __mod__, __divmod__, __pow__, __lshift__, __rshift__, __and__, __or__, __xor__

MAGIC METHODS

- __str__ vs __repr__
    - __str__ is to be human readable (more for clients)
    - __repr__ is to be unambiguous (more for developers)
    - __repr__ is backup for __str__
- __call__
    - Allows the class instance to be called as a function
- __new__ is the constructor
- __init__ is the initializer

OOP EXERCISES

- Create class *PublicPlace*.
    - Instance attributes initialised on init: city(str), address(str)
    - Instance method get_address(): returns city concatenated with address
- Create class *Restaurant*, that inherits *PublicPlace* class.
    - Instance attributes are: category(str), rating(int), city(str), address(str)
    - Init will overwrite parent init method
    - Class attribute: count

- Print how many *Restaurant* objects were created (use class attribute *count*)
- Print the details of each *Restaurant* object (implement __*str*__)
- Compare two *Restaurant* objects. The object that has the higher *rating* will be greater.

PYTHON CLASS TEMPLATE: METACLASS

- The class of a class
- Default metaclass is ‘type’