Pycon Tutorial - Python 101

Simon Lau

18 Jun 2014

1. Administration
2. Session 1
3. Tea Break
4. Session 2

Administration

Internet and downloads

Internet Access

SSID: PyCon_SG

No Password

Slides

http://goo.gl/U7SMVX

Author

Simon Lau

Lecturer

School of Infocomm

Republic Polytechnic

simon_lau@rp.edu.sg

Agenda

- Session 1
  - Intro
  - Numbers and Operators
  - Strings
  - Lists and Tuples
- Tea Break
- Session 2
  - Blocks and Loops
  - Dictionaries
  - Classes and Objects

What other programming languages have you learnt before?

Session 1

Intro

What is Python?

- Programming scripting language
  - Runs on Windows, Mac OS X, Linux
- Developed by Guido van Rossum
- Released in 1994
- Open Source
- Current version `2.7.7` & `3.4.1`

What is it used for? (Part 1)

- Web Development
  - Django (Python Web Framework)
  - Google (search spiders)
  - Yahoo (maps application)
  - Facebook (Tornado web server framework)
  - Youtube (almost whole website)

What is it used for? (Part 2)

- Games
  - AI Programming
  - Team balancing
  - Score keeping
- Graphics / Rendering
  - Industrial Light & Magic
  - Blender 3D

What is it used for? (Part 3)

- Financial
  - ABN AMRO Bank
- Scientific
  - National Weather Centre, US
  - NASA
- Educational
  - Rice University
  - Republic Polytechnic

Why named 'Python'?

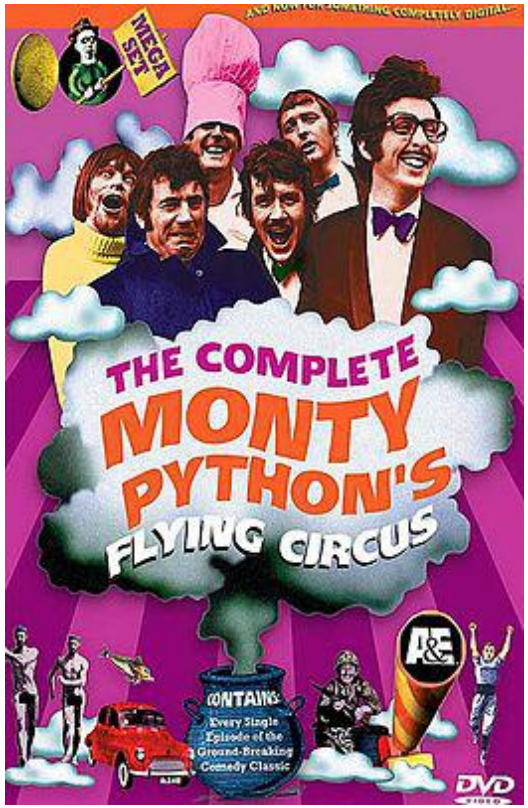

Figure : Named after Monty Python's Flying Circus

Using Python

- Can be downloaded and installed locally
  - `http://python.org/download/`
- We can also use it online
  - `http://www.pythonfiddle.com/`

Hello World

Python 2.x

```
print "Hello World"
```

Python 3.x

```
print("Hello World")
```

Numbers and Operators

Numeric Types (Built-in)

```
a = 3          # int
b = 3.5        # float
```

Let's do some calculation

```
print 13 + 6
print 2 * 12
print 20 / 4
print 22 / 4
print 22 % 4
print 3 ** 4
```

Available calculations

Command	Name	Example	Output
+	Addition	`3 + 4`	7
-	Subtraction	`9 - 5`	4
*	Multiplication	`3 * 4`	12
/	Division	`16 / 3`	5
%	Reminder	`16 % 3`	1
**	Exponent	`2 ** 3`	8

Integers vs. Long vs. Floats

`int` are whole numbers (`32` bit)

```
8 4 5683 0 -312 2147483647
```

`long` are whole numbers (unlimited precision)

`float` are floating point numbers, decimals

```
2.3 14.4532 -301.0 3E14
```

Other Operations

Operation	Details
`abs(x)`	Absolute value or magnitude of `x`
`int(x)`	`x` converted to `int`
`long(x)`	`x` converted to `long`
`float(x)`	`x` converted to `float`
`divmod(x, y)`	the pair `(x // y, x % y)`
`round(x[, n])`	`x` rounded to `n` digits (`n` defaults to `0`)
`math.floor(x)`	the greatest integral float `<= x`
`math.ceil(x)`	the least integral float `>= x`

Order of calculation

- General mathematical rules apply:
  - Power, square root
  - Multiplication, division
  - Plus, minus
- Can use brackets in calculation

```
3 * (4 + 5) ** 2
```

Introducing variables

- Python does not have variable declaration
- Do not need to indicate a type

```
a = 8
b = 12
print a * b
```

Quack Quack

- Does Python have data type?
- Python uses Duck Typing[1]

when I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.

[1] http://en.wikipedia.org/wiki/Duck_typing

Type() function

```
a = 8
b = 8.0
```

Use the `type()` function

```
print type(a)
print type(b)
```

Swap Exercise

After assigning values to 2 variables like this

```
a = 8
b = 20
```

- How do swap the values of the variables
  - make value of `a` to be 20
  - make value of `b` to be 8

Solution

Classic

```
a = 8
b = 20
c = a
a = b
b = c
print a, b # 20 8
```

Use tuple unpacking

```
a = 8
b = 20
a, b = b, a
print a, b # 20, 8
```

Strings

Strings

`str` are text-based variables which can be defined in a number of ways:

```
a = "hello"
b = 'hi'
c = """hello
all"""
```

String operators

Just like numbers, `str` can be added

```
a = "Hello"
b = "world!"
print a + b
```

- Is it possible to multiply `2 str`?
- Is it possible to multiply an `int` and a `str`?

Solution

- `int * str`?
  - Yes, this one is possible
- `str * str`?
  - No, multiplying `str` is not possible
- `float * str`?
  - No, not possible either

```
print 3 * 'apple'       # 'appleappleapple'
print "apple" * 'pear'
print 2.5 * 'apple'
```

Difference between + and ,

```
a = 'Hello'
b = 'world!'
print a + b # 'Helloworld!'
print a, b  # 'Hello world!'
```

In the above example, the difference between using `print` and not using `print` becomes clear. In normal programs you need to use `print` to display output to the screen.

String methods: capitalize()

- A complete list of `str` methods can be found here:
  - `http://docs.python.org/library/stdtypes.html#string-methods`
- We'll look at couple of them:
  - `str.capitalize()`:
    - `return` a copy of the `str` with only its first character capitalized

```
mystring = "hello"
print mystring.capitalize()
print mystring
mystring = mystring.capitalize()
print mystring
```

String methods: count()

- `str.count(sub[, start[, end]])`:
  - return the number of non-overlapping occurrences of substring sub in the range `[start, end]`

```
mystring  = "mississippi"
print mystring.count('i')          # 4
print mystring.count('is')         # 2
print mystring.count('is', 4)      # 1
print mystring.count('i', 0, 7)    # 2
```

String methods: count() syntax

The `start` and `end` of the count are optional, `start` is included, `end` is excluded.

m	i	s	s	i	s	s	i	p	p	i
0	1	2	3	4	5	6	7	8	9	10

Figure : Index of String

String methods: isalnum()

- `str.isalnum()`:
  - `return True` if all characters in the `str` are alphanumeric and there is at least one character
  - `return False` otherwise

```
a = "KL838"
print a.isalnum()   # True
b = "KL838!"
print b.isalnum()   # False
c = "838"
print c.isalnum()   # True
d = "KL"
print d.isalnum()   # True
```

String Methods Exercise

Which method can you use to get a `str` in all uppercase?

Which method can you use to replace all instances of `'a'` with `'A'` in a `str`?

`http://docs.python.org/library/stdtypes.html#string-methods`

Solution: String.upper()

- Which method can you use to return the `str` in all uppercase?
- `upper()`:
  - returns a copy of the `str` converted to uppercase

```
river = 'mississippi'
print river.upper()
```

Solution:

- Which method can you use to replace all instances of `'a'` with `'A'` in a `str`
- `str.replace(old, new[, count])`
  - return a copy of the `str` with all occurrences of substring `old` replaced by `new`
  - If the optional argument `count` is given, only the first `count` occurrences are replaced.

```
word = 'abracadabra'
print word.replace('a', 'A')
print word.replace('a', 'A', 2)
```

Substrings

- To retrieve a part of a `str` square brackets are used
- For example, to retrieve the `4` first characters of *'Mississippi'* you can do the following

```
word = 'mississippi'
print word[0:4]
```

Again the `start` is included, `end` is excluded.

Substrings exercise

- How do you extract the `'ssiss'` from the word `'Mississippi'`
- How do you extract the last `4` characters from the word `'Mississippi'`?

Substring solution 1

- First character is position 2
- Last character is position 6

m	i	s	s	i	s	s	i	p	p	i
0	1	2	3	4	5	6	7	8	9	10

Figure : Index of String

```
word = 'mississippi'
print word[2:7]
```

Substring solution 2

m	i	s	s	i	s	s	i	p	p	i
0	1	2	3	4	5	6	7	8	9	10

Figure : Index of String

```
word = 'mississippi'
print word[7:11]
print word[-4:11]
print word[-4:]
```

Immutability

`str` are immutable, which means they cannot be changed

When changing a `str`, assign the value to a new `str`

```
word = 'mississippi'
print word[0]
newstr = 'x' + word[1:]
word[0] = 'x'       # Error
```

Boolean

- `bool` values are either `True` or `False`

```
x = True
y = 5  > 9
print y      # False
```

- We'll cover more `bool` values when we discuss conditionals later

Lists and Tuples

Lists

- `list` are a one of Python's strong points, they are flexible and easy to use
- A `list` is a container that holds a number of other objects, in a given order
- Items can be added and removed from the `list`
- To create an empty `list`, use the following code

```
mylist = []
print mylist    # []
```

List elements

- Like letters in a `str`, elements in the `list` can be referenced using an index
- The first element in a `list` , has index `0`, the second `1`, and so on
- Square brackets are used for this

```
newlist = ['a', 'b', 'c', 'd']
print newlist[1]      # 'b'
```

List Exercise

- How do you get the last element from the `list`?
- How do you get the second-last element from the `list`?
- How do you get the first `3` elements from the `list`?

List Solution

How do you get the last element from the `list`?

```
newlist = ['a', 'b', 'c', 'd']
print newlist[-1]      # 'd'
```

How do you get the second-last element from the `list`?

```
print newlist[-2]      # 'c'
```

How do you get the first 3 elements from the `list`?

```
print newlist[:3]      # ['a', 'b', 'c']
```

List methods

Adding items to a `list` can be done with `list.append(item)`

```
mylist = []
mylist.append('durian')
mylist.append(1)
mylist.append('apple')
print mylist                    # ['durian', 1, 'apple']
```

Other ways to add items

`list.extend(sequence)` and `list.insert(index, item)`

```
mylist = ['durian', 1, 'apple']
mylist.extend(['banana', 5])
print mylist
# ['durian', 1, 'apple', 'banana', 5]
mylist.insert(2, 'starfruit')
print mylist
# ['durian', 1, 'starfruit', 'apple', 'banana', 5]
```

Removing items: list.remove(item)

- `list.remove(x)`:
  - removes the first item from the `list` whose value is `x`
  - `return` an error if there is no such item

```
mylist = ['durian', 1, 'starfruit', 'apple', 'banana', 5]
mylist.remove('durian')
print mylist       # [1, 'starfruit', 'apple', 'banana', 5]
mylist.remove(5)
print mylist       # [1, 'starfruit', 'apple', 'banana']
```

Removing items: list.pop()

- `list.pop([i])`:
  - removes the element at position `i`
  - `if i` is not given, then the last element will be removed
  - The function `return` the element that is removed.

```
mylist = [1, 'starfruit', 'apple', 'banana']
print mylist.pop()      # 'banana'
print mylist.pop(0)     # 1
print mylist            # ['starfruit', 'apple']
```

Other list methods

Some of the other useful methods are:

`list.count(x)` return the number of times `x` appears in the `list`
`list.sort()` sort the items of the `list`
`list.reverse()` reverse the elements of the `list`

List functions

There are 2 useful functions you can use for `list`:

`len(list)` `return` the number of elements in the `list`

`sum(list)` `return` the `sum` of the numbers in the `list`, produces an error when not `all` elements are numbers

Creating lists using range

- You can create `list` manually
- You can also creates `list` using functions, like `range(start, end, step)`

```
mylist1 = range(1, 10)
print mylist1
mylist2 = range(0, 10)
print mylist2
mylist3 = range(0, 10, 2)
print mylist3
```

List exercise

- How can you create a `list` like `[0, 3, 6, 9, 12]`?
- How can you create a `list` like `[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`?
- How can you create a `list` like `[4, 2, 0, -2, -4]`?

List solution

- How can you create a `list` like `[0, 3, 6, 9, 12]`?

```
print range(0, 13, 3)
```

- How can you create a `list` like `[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`?

```
print range(10, 0, -1)
```

- How can you create a `list` like `[4, 2, 0, -2, -4]`?

```
print range(4, -5, -2)
```

Tuples

- `tuple` are immutable `list`
- Instead of square brackets `[]`, round brackets `()` are used

```
newtuple = (1, 2, 'a', 1, '1')
print newtuple
```

Tuples Exercise

- Can you remove items from the `tuple`?
- Can you add new items to the `tuple`?
- What about sorting the `tuple`?

Tuples Solution

- **No**, **No** and **No**
- `tuple` is immutable, so cannot be changed
- But, you can convert the `tuple` to a `list`, edit, and convert back to `tuple`

```
tup = (1, 2, 3)
lis = list(tup)
lis.remove(2)
tup = tuple(lis)
print(tup)      # (1, 3)
```

Tea Break

Session 2

Blocks and Loops

Blocks and Loops

- Python does not demark blocks with brackets {}
- It uses indentation to demark blocks
- Let's look at a `for` loop

```
for item in list:
    print something
```

Blocks exercise

- Can you print out each on a new line, the elements of the following `list`
  - `['Hello', 'we', 'are', 'learning', 'Python']`
- How can we get them all on the same line, instead of a new line for them all?

Blocks solution

```
list = ['Hello', 'we', 'are', 'learning', 'Python']
for word in list:
    print word
"""
Hello
we
are
learning
Python
"""
for word in list:
    print word,
'Hello we are learning Python'
```

Loop Exercise

- To do a classic Java `for` loop like

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

- You can use the `range` function
- Can you write a loop which shows the exponentials of 2?

```
# 1, 2, 4, 8, ... 256
```

Loop Solution

```
for i in range(9):
    print 2 ** i

"""
1
2
4
8
16
32
64
128
256
"""
```

Multiplication Table Exercise

Write a small program that will print the times table for 5

$1 \times 5 = 5$

$2 \times 5 = 10$

...

$10 \times 5 = 50$

Multiplication Table Solution

```
num = 5
for i in range(1, 11):
    r = i * num
    print str(i) + ' x ' + str(num) + ' = ' + str(r)
```

User Input

- To make more interesting programs, you need user input
- Use the `raw_input(prompt)` function

```
who = raw_input('Who are you?')
print who
```

User Input Exercise

What will the following code do?

```
x = raw_input('Gimme a number:') # Enters 28
print x * 10
```

User Input Solution

```
28282828282828282828
```

```
raw_input() return a str
```

Dictionaries

Dictionaries

- Related to `list` are `dict`, which in other languages are called associative arrays
- These `dict` contain key, value pairs
- To create a `dict`, do the following

```
telnos = {'John': 1234, 'James':3456}
```

Key	Value
'John'	1234
'James'	3456

Figure : Creates the above dictionary

Adding to / Removing from a dictionary

To add a new key-value pair do the following

```
telnos = {'John': 1234, 'James':3456}
telnos['Jack = 9876']
print telnos
```

To remove a pair from the `dict`, for example James' telephone number, use the `del` function, like this

```
telnos = {'John': 1234, 'James':3456, 'Jack': 9876}
del telnos['James']
print telnos
```

Looping through dictionaries

To loop through all values in the `dict`, you can use a slightly modified `for` loop, in combination with the `iteritems()` method

```
telnos = {'John': 1234, 'Jack': 9876}
for k, v in telnos.iteritems():
    print k, v
```

Conditionals

`if` statements are pretty straight forward in Python, they have the following format

```
a = 10
if a > 0:
    print 'A is bigger than zero'
else:
    print 'A is smaller than or equal to zero'
```

So, again there are no brackets, but indentation is used to demark blocks.

Else statements

```
a = 10
if a > 0:
    print 'A is bigger than zero'
elif a < 0:
    print 'A is smaller than zero'
else:
    print 'A is equal to zero'
```

Conditional Exercise

- Can you write a program that
  - reads an `input` from the user
  - `print 'letters', 'digits'` or `'mixed'` based on the input?

Conditional Solution

```
text = raw_input('Give me something:')
if text.isalpha():
    print 'All letters'
elif text.isdigit():
    print 'All digit'
elif test.isalnum():
    print 'Mixed'
else:
    print 'Invalid input'
```

Other comparisons

Operator	Meaning
<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal
`is`	object identity
`is not`	negated object identity

Defining functions

- Functions in Python are declared using `def`
- For example we can create a function to add up three numbers using the following code

```
def sum3(a, b, c):
    return a + b + c
```

Function Exercise

- Can you make function that
  - Takes a `list` of numbers as a parameter
  - Returns the average value of the `list` items?

Function Solution

```
def avg(mylist):
  return sum(mylist) / float(len(mylist))

print avg([1, 2, 3, 4])
```

Classes and Objects

Simple Classes

- Python is mostly object oriented
  - `int float list` etc are all objects
- Creating own `class` including properties, methods and constructors is fairly straight forward
- We'll just cover values, getters, setters, instance and constructor

Creating a class: properties

Creating an empty `class`

```
class Employee:
    pass
```

This code works but is uninteresting as an empty `class`. So we'll add some properties like

```
class Employee:
    name = ''
    salary = 100
```

- The `class` Employee now has two properties:
  - `name`, which is a `str` with initial value `''` (empty string)
  - `salary`, which is an `int` with initial value `100`

Creating a class: methods

```
class Employee:
    name = ''
    salary = 100

    def get_salary(self):
        return self.salary

    def get_name(self):
        return self.name

    def set_name(self, name):
        self.name = name

    def set_salary(self, salary):
        self.salary = salary
```

Creating a class: self

- This needs some explanation, let's start with `get_salary(self)` method
- A method is a function in a `class`, so we use the `def` like we used in creating normal functions
- After `def` we write the function name (i.e. `get_salary`)
- This method has one parameter: `self`. This is a default parameter `all` methods in a `class` have

Instantiating a class (I)

- Using the `class`, by creating a new object, goes like this:

```
emp1 = Employee()
```

- What is the name of Employee `emp1`?
- What is the salary of Employee `emp1`?

Creating a class: constructor

- A constructor is a special method that automatically gets called when a new object is created and is used to initialize the values in the object.
- Creating a constructor in Python goes as follows

```
def __init__(self, name, salary):
    self.name = name
    self.salary = salary
```

Instantiating a class (II)

Now creating a new employee goes as follows:

```
emp1 = Employee('John', 100)
```

Now, the name of `emp1` is *'John'* and his salary is 100

Creating a class: Constructor parameters

- Default values to the parameters can be given like this

```
def __init__(self, name='John', salary=200):
    self.name = name
    self.salary = salary
```

Classes Exercise

After we use our `class` and the following code

```
emp1 = Employee()
emp2 = Employee('Guido')
emp3 = Employee('John', 100)
emp4 = Employee(500)
```

What are the salaries and names of each of the 4 employees we created?

Classes Exercise

- emp1:
  - Name: *'John'*
  - Salary: 200
- emp2:
  - Name: *'Guido'*
  - Salary: 200
- emp3:
  - Name: *'John'*
  - Salary: 100
- emp4:
  - Name: *'500'*
  - Salary: 200

Method Exercise

Can you add a method to the Employee `class` that will increase the salary of the employee with a certain percentage?

Method Solution

- Pass the percentage as parameter
- Convert it to `float`
- Multiply the `self.salary` with `1 + percentage / 100`

```
def give_raise(self, percent):
    self.salary *= (1 + float(percent) / 100)
```

Exception Handling

- Exceptions are used to cope with unpredicted situations.
- For example, you are doing a calculation with some variables and you divide by 0
- Without exceptions your program would crash
- But with using exceptions your program can still work and handle the error as you wish.

Exception Example

```
for i in range(-5, 5):
    print 100 / i
```

Will result in

```
-20
-25
-34
-50
-100
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
ZeroDivisionError: integer division or modulo by zero
```

Using Exceptions

```
try:
    for i in range(-5, 5):
        print 100 / i
except:
    print 'Ooops something is not correct'
print 'After which we can still do other stuff'
print 'Like ask the user for input'
again = raw_input('You want to try again?')
```

Final Exercise

- Can you create a simple game where a user can play hangman
- The computer will show the number of characters from a word randomly chosen from a `list`
- The user will enter a letter
- If the letter occurs in the word the positions of that letter will be revealed
- If the letter does not occur in the word one part of the gallows will be built
- After `x` number of wrong guesses or when the word is discovered the game is over

Thank You