An Introduction to Python

MACbioIDi – February – March 2018

Carlos Luque

Outline

- Introduction
- Syntax and semantics
  - Statements
  - Blocks
  - Reserved words
  - Variables
  - Operations
  - Flow Control
  - Functions

Outline

- Classes
- Modules
- Input and output
- Errors and Exceptions

Introduction

- An interpreted high-level programming language for general-purpose programming
- Design philosophy
  - Code readability
  - Simple syntax
- There are plenty of standard libraries
  - Extend functionalities

Introduction

- Multiple Programming Paradigms
  - Imperative
  - Functional
  - Procedural programming
    - Blocks and Scopes
  - Object-oriented programming
    - Objects
- Python have two syntax versions:
  - 2.7: This presentation
  - 3.X: New syntax

Statement

- Each statement is defined in a logic line
- Each logic line is declared
  - One or more physical lines (\)
  - End: carriage return
- Comment a statement with “#”

One statement	A = 1 + 2
One statement (multiple lines)	B = 20 + \ 18
Comment	# comment line

Blocks

- A block has
  - One statement or more statement/s
- Blocks are delimited by **indentation**
  - White space indentation
    - No curly bracket , {}
  - Each increasing of the indentation
    - New block is generated
  - Each decreasing of the indentation
    - End the current block

Off-side rule

Blocks

- Several examples

```
statement
    statement
    statement
    statement
    .....
statement
statement
```

```
    statement
    statement
    statement
    .....
statement
statement
```

Wrong:
The indentation cannot be applied in first block

Reserved words

- No use in variables, special meaning

and	assert	break	class	continue
def	del	elif	else	except
exec	finally	for	from	global
if	import	in	is	lambda
not	or	pass	print	raise
return	try	while		

Variables

- Context-sensitive
  - The variable “B” is different than variable “b”
- Declaration:
  - Begin: a letter or dash (_)
  - Next: Zero or more alphanumeric
  - __XXX__: special meaning

Variab	Right
v3425	Right
02Var	Wrong
__member	Right

- Untyped variable names
  - The type of a variable is not defined when the variable is declared

Variables

- The type is checked in runtime
  - Duck typing
- Several primitive types
  - Numeric types
    - Bytes, integer, long integer, float, complex
  - Sequence types
    - String, list, tuple, dictionary, bitarray
  - Grouping
    - Set
  - Boolean

Variables: Numeric

Type	Mutable	Example
Integer	No	3 03 in Octal 0x3 in Hexadecimal
Long integer	No	25L 03L in Octal 0x3L in Hexadecimal
Float	No	2.3
Complex	No	2+3j 2.3 + 0.5j 2L + 9j 2L + 9Lj is wrong

Variables: Sequence

Type	Mutable	Attribute	Example
String	No	• Quotation marks (' ‘, " ") • May several lines	“Hello” ‘Hello’ “Hello Marilola”
List	Yes	Can contain mixed types	[1, 3.2, 2+2j, “Juan”]
Typle	No	• Can contain mixed types •__add__ appends new element	(1, 3.2, 2+2j, “Juan”) (“juan”, 6).__add__((1,)) gives (“juan”, 6, 1)
Dictionary	Yes	• List with key and value pairs • Can contain mixed types	{2:"e", 1:"h", 4:"g"}
Bytearray	Yes		bytearray([1,2,3])

Variables: Set and boolean

- Group

Type	Mutable	Attribute	Example
Set	Yes	• No duplicates • Can contain mixed types	{ 4.0, 1, True} {1,1,1} = {1}

- Boolean

Type	Mutable	Attribute	Example
bool	No	• True or False	A = True

Variable

- Know type of a variable
  - type(X) X = variable
- Check type of a variable
  - isinstance(X, type) X = variable
- Delete
  - del(X), del(X;X;…) X = variable

Operations

- Assignment
  - “=”
- Numeric
  - Unary: “++X, --X”
  - Binary: “+, -, *, **, /, //, %, round”
    - // = floor division
  - Bits: “<<, >>, ~, |, ^, &”
    - | = OR, ^ = XOR, & = AND
  - They can be joined to the assignment “=”
    - Binary: “+=, -=, *=, **=, /=, //=, %=”
    - Bits: “<<=, >>=, ~=, |=, ^=”

Operations

- Comparison
  - “<, >, ==, <=, >=, != or <>”
- Logic (boolean)
  - “and, or, not”
- Test identity
  - “is, is not”
- Test Belonging
  - “in, in not”

Assignment

- Special cases

a = b = c = 5

It is the same as
a = 5
b = 5
c = 5

Unpacking assignment

a , b , c = 5,7,9
a , b , c = (5,7,9)
a , b , c = [5,7,9]

It is the same as
a = 5
b = 7
c = 9

Operations in Sequence

- Indexation
  - sequence[index] when
    - If L is the number of the elements in the sequence
      - index = [-L … 0 …. L-1]
        - » negative values is reverse order
- Division
  - sequence[start: stop :strive]
    - start, stop, strive are optional
    - The default value of strive is one
    - If L is the number of the elements in the sequence
      - start, stop is in [-L …. 0 …… L-1]
        - » negative values is reverse order

Operations in Sequence

- Number of the elements
  - "Len(sequence)"
- Concatenation (the same type of sequence)
  - "Seq1 + Seq2" Seq1 and Seq2 are sequence
- Repetition (the same type of sequence)
  - "X*n" n = X times n
    - If n <= 0, sequence is empty

Flow control

- Conditional

```
if condition :
        block
[elif expression :
        block]
[else:
        block]
```

[] is optional

Condition is True or False

- Loop
  - For

```
for  target in Iterable :
        block
```

Iterable is Strings, Lists, Tuples, Dictionaries, Group s

  - While

```
while  condition :
        block
```

Flow control

- Loop
  - Break
    - Break the loop
  - Continue
    - Run next loop
  - Range(X)
    - Generates a list from 0 until X

Flow control: Special condition

- Special condition
  - True if
    - Number > 0
    - No empty in
      - String, List, Tuple, Dictionary, Group
  - False if
    - Number = 0
    - Empty (None) in
      - String, List, Tuple, Dictionary, Group

Functions

- The syntax

```
def function-name(parameters) :
    block
```

- Block may have zero or more statement(s) “return”
- Parameters
  - Zero or more variables separating with comma“,”
  - The variables may be initialized with an assignment

```
def sum(b, c = 1) :   # The variable c is initialized to 1
    return b + c
```

Functions

- Invoking a defined function

  function-name(parameters)

  – If the function has default value in its parameters, it may be invoked omitting those parameters

Def increment (operatorA, opertatorB = 1):
  return opertatorA + operatorB

It is invoked in three ways:
- increment(5)
  - returned value is 6
- increment(5, 3)
  - returned value is 8
- increment(5,opertatorB = 4)
  - returned value is 9

Functions

- The variables are used in local scope of a function
- If a function only defines a statement “returns expression” and it is used once time, it can be changed to a function lambda

lambda parameters **:** expression

Object-Oriented Programming

- Python supports
  - Instantiation (instance)
    - Class
    - Object
  - Inheritance
    - A class inherits another class

Classes

- Syntax

  class className (inherited classes) **:**
  block

- Block can be defined
  - Members
    - No private
  - Methods

    def methodName (self, parameters) **:** block
  - Constructor for a class and initializes the members

    __init__ (self, parameters): block

Important: [1] The first parameter of the class methods is always referenced to the current instance of the class. By convention, this argument is always named “self”

Classes

- Examples:

```
class vehicle ():
    SeatNumber = -1

    def __init__ (self,  seats):
        self.SeatNumber = seats

    def getSeats(self):
        print self.SeatNumber
```

```
class land(vehicle):
    WheelNumber = -1

    def __init__(self, seats, wheels):
        vehicle.__init__(self, seats)
        self.WheelNumber = wheels

    def getWheels(self):
        print self.WheelNumber
```

```
class car(land):
    def __init__(self, seats):
        land.__init__(self, seats, 4)
```

Instance

- Instance of a object is created invoking the className as a function
  - className(parameters)
- The function “isinstance”, is a special function that checks to see if an instance is an instance of a certain type of class

isinstance (instance, TypeClass)

Instance

- Examples: two cars

```
carToyota = car(3, "Toyota")
carToyota.getBrand()
carToyota.getSeats()
```

```
carSeat = car(5,  “Seat")
carToyota.getBrand()
carToyota.getSeats()
```

Modules

- A module is a file containing Python definitions and statements.
- The file name is the module name with the suffix .py appended
- Modules can import other modules via the statement “import”

Modules

- There are several ways to import one or more modules
  - **import** moduleName
    - import a whole module
  - **import** moduleName**,** moduleName**,** ….
    - import several whole module
  - **import** moduleName **from** classname(s) or function(s)
    - import some parts of a module
  - **import** moduleName **as** otherName
    - import a whole module and create an alia

Input and Output

- Output
  - **Print**: evaluates each expression in turn and writes the resulting object to standard output
    - If objects is not string, they are firstly converted to strings
  - Converting any value to a string
    - **str**: in fairly human-readable
    - **repr**: in interpreter-readable

Input and Output in File

- Open a file
  - Function “open” **open**(filename, mode)
    - Mode
      - w: writing
      - a: append at the end of a file
      - r: reading
      - r+: reading and writing
- Close
  - Function “close()”

Input and Output in File

- Read a file
  - Functions “read” and “readline”. Both return a string
- Writing
  - Functions “write”
- Example open the file ‘data.csv’, reading a line, writing ‘example’ and close the file

```
f = open(‘data.csv’, r+)
line  = f.readline()
print line
f.write(‘example’)
f.close()
```

Input and Output in File

- [2] It is good practice to use the ‘with’ keyword when dealing with file objects. This has the advantage that the file is properly closed after its suite finishes, even if an exception is raised on the way. It is also much shorter than writing equivalent ‘try-finally’ blocks:

```
with open('workfile', 'r') as f:
    read_data = f.read()
f.closed
```

Error and Exceptions

- A exception is an error in runtime
- To handle exceptions, several options

Option A

```
Try
    block may generate an exception
except  exception type:
    handled block
```

Option B

```
Try
    block may generate an exception
except  exception type:
    handled block
else
    not exception code that must be executed
```

- The function ‘raise’ generates an exception

References

- [1] https://pythontips.com/2013/08/07/the-self-variable-in-python-explained/
- [2]https://docs.python.org/2/tutorial/
- https://docs.python.org/3/tutorial/

An Introduction to Python