Python Training

Lesson 01: A basic overview

Table of Contents

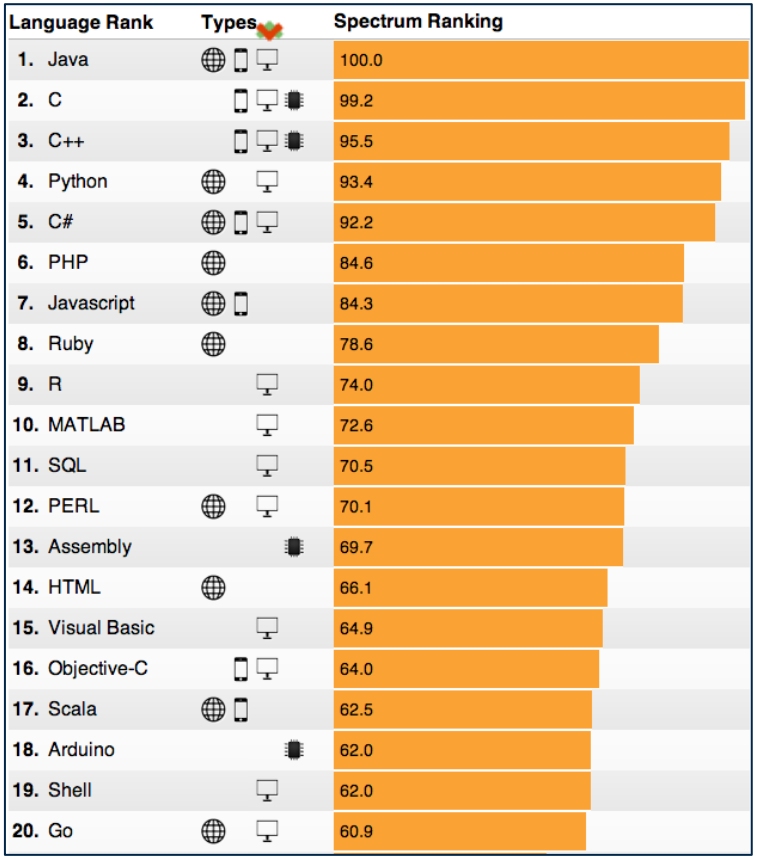
Introduction to Python

- What is Python
- History
- Features of Python
- Installing Python & Online Documentation
- Running Python
  - Using interpreter/cli
  - Setting up IDE
- Python 2 vs Python 3

What is Python?

- Python is a
  - **High level programming language**
  - **Interpreted**
  - **Interactive**
  - **Object Oriented**
- Python development started in December 1989. Designed by and principal author Guido van Rossum
- Influences from other languages
  - **ABC : Core syntax directly inherited**
  - **Bourne shell : Interactive Interpreter**
  - **Lisp & Haskell : Features such as list comprehensions, map functions**
  - **Perl : Regular expressions, shell script**

Top programming languages 2014-15



IEEE Spectrum

Feb 2014	Feb 2015	Language	Ratings	Change
1	1	C	16.488%	-1.85%
2	2	Java	15.345%	-1.97%
3	4	C++	6.612%	-0.28%
4	3	Objective-C	6.024%	-5.32%
5	5	C#	5.738%	-0.71%
6	9	JavaScript	3.514%	+1.58%
7	6	PHP	3.170%	-1.05%
8	8	Python	2.882%	+0.72%
9	10	Visual Basic .NET	2.026%	+0.23%
10	-	Visual Basic	1.718%	+1.72%

TIOBE Index

Features of Python

- Feature highlights include:
  - **Easy-to-learn**: Python has relatively few keywords, simple structure, and a clearly defined syntax.
  - **Easy-to-read**: Python code is clearly defined and if well written visually simple to read and understand.
  - **Easy-to-maintain**: Python's success is that its source code is fairly easy-to-maintain.
  - **A broad standard library**: One of Python's greatest strengths is the bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
  - **Interactive Mode**: Support for an interactive mode in which you can enter results from a terminal right to the language, allowing interactive testing and debugging of snippets of code.

Features of Python

- Feature highlights include:
  - **Portable**: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
  - **Extendable**: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
  - **Database Aware**: Python provides interfaces to all major commercial databases.
  - **GUI Programming**: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
  - **CGI Programming**: Supports server and client side scripting, many libraries and modules
  - **Scalable**: Python provides a better structure and support for large programs than shell scripting.

Features of Python

- Important structural features that make it an efficient programming tool:
  - Built-in high level data types: strings, lists, dictionaries, etc.
  - The usual control structures if, if-else, if-elif-else, while loop, (a very powerful) for loop.
  - It can be used as a scripting language or can be compiled to byte-code for building large applications. (Using third party tools such as Py2exe or Pyinstaller, Python code can be packaged into standalone executable programs)
  - Supports automatic garbage collection.
  - It can be easily integrated with Fortran, C, C++, CORBA, and Java, etc…

Installing Python and documentation

- Getting Python:
  - The most up-to-date and current source code, binaries, documentation, news, etc. is available at the official website of Python: http://www.python.org/
- Documentation
  - You can download the Python documentation from the following site. The documentation is available in HTML, PDF, and PostScript formats: http://docs.python.org/index.html
- Tutorial
  - You should definitely check out the tutorial on the Internet at: http://docs.python.org/tutorial/.

Installing/Running Python IDE

Demo

Language syntax

- Structure of python code
- Variables & Data Types
- Operators

Simple python code

- Below is a simple python program on a windows environment.
- Quick highlights of syntax
  - Comments begin with a hash sign (#)
  - semicolon not mandatory, only to combine multiple statements
  - Blocks of code called **suites** are denoted by line indentation (no curly braces!!)
  - Variables are auto typed, no need to be declared

```
# This is a comment
x='';
x=input('Enter your name:')
print ('Hello ',x)
x=input('Enter your age:')
y=float(x)
x=int(y)
if x < 0 or x > 150:
  print("invalid entry!")
else :
  if x > 17 :
    print("You are eligible to vote")
  else :
    print("you are not eligible to vote")
```

Variables

- No need to declare
- Need to assign (initialize)
  - use of uninitialized variable raises exception
- Auto typed

```
if friendly: greeting = "hello world"
else: greeting = 12**2
print greeting
```

- Variable names:
  - can contain both letters and digits, but they have to begin with a letter or an underscore.
  - Punctuation characters such as @, $, and % are not allowed.
  - Are case sensitive.
  - Cannot be any of the keywords

Reserved words

- Python Reserved words:
  The following list shows the reserved words in Python. These reserved words not to be used as constant or variable or any other identifier names

and	exec	not	as
assert	finally	or	nonlocal
break	for	pass	True
class	from	print	False
continue	global	raise	None
def	if	return	
del	import	try	
elif	in	while	
else	is	with	
except	lambda	yield	

Basic Data Types

Numbers

- Operators
- Functions

Boolean

- Operators

Strings

- Operators
- Functions

Numbers

- Python supports four different numerical types:
  - **int** (signed integers) = C long precision
  - **long** (long integers [can also be represented in octal and hexadecimal]) unlimited precision
  - **float** (floating point real values) = C double precision
  - **complex** (complex numbers) = C double precision
- They are immutable data types
- Examples:

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j

Numbers: Operators

- Arithmetic operators:

Operator	Description
+	Addition - Adds values on either side of the operator
-	Subtraction - Subtracts right hand operand from left hand operand
*	Multiplication - Multiplies values on either side of the operator
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder
**	Exponent - Performs exponential (power) calculation on operators
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.

- Note: No ++ -- operators available
- Note that Integer division will produce truncated result
  - Eg: >>> 1//2 will produce 0
  - Workaround: 1./2 or float(1)/2

Bitwise operators

- Bitwise operators

Operator	Description
~	Bitwise complement, unary operator
&	Bitwise ANDing, binary operator
|	Bitwise Oring, binary operator
^	Bitwise XORing, binary operator
<<	Left shift, will add trailing zeros
>>	Right shift, will add leading zeros

- Example: 7<<2, a&b, a|b, 6^8, ~7
- Note: These won't work on float/complex data types

Numbers: Functions

- Internally each of the objects have functions , e.g. as_integer_ratio, numerator, denominator etc

- Support available also from "math" module
  - The math module contains the kinds of mathematical functions you'd typically find on your calculator.
  - Comes bundled with default installation.

```
>>> import math
>>> math.pi # Constant pi
3.141592653589793
>>> math.e # Constant natural log base
2.718281828459045
>>> math.sqrt(2.0) # Square root function
1.4142135623730951
>>> math.radians(90) # Convert 90 degrees to radians 1.5707963267948966
```

Boolean

- Python supports *bool* data type with values True and False
- Relational operators applicable as below:

Operator	Description
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

- Logical Operators: and , or, not

String

- Strings in python are immutable.
- You can visualize them as an immutable list of characters.

a =	H	E	L	L	O
	0	1	2	3	4
	-5	-4	-3	-2	-1

- You can use single quotes, doubles quotes, triple quotes (multiline string) and “r” (raw string)
  - str1 = “Hello World!”
  - str2 = “You can’t see me”
- Python has many built-in functions to operate on strings.
- Usual list operations like +, *, splice, len work similarly on strings.

String

- Some helpful functions
  - find() : finds a substring in a string
  - split() : very useful when parsing logs etc.
  - format() : A very powerful formatting function that uses a template string containing place holders. Refer documentation for completeness
    - s2 = "I am {1} and I am {0} years old.".format(10, “Alice”)

- The in and not in operators test for membership

```
>>> "p" in "apple"
 True
>>> "i" in "apple"
 False
>>> "x" not in "apple"
 True
```

Data Structures

- List
- Tuple
- Dictionary
- Set
- Data Type Conversion

List

- A list is an ordered collection of values.
- Similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

```
a = [] #Empty list
b = [10, 20.1, "ABC"] #List with different data types
nested = ["hello", 2.0, 5, [10, 20]] #Nested List
print b[0]
print nested[3][1]
```

- Accessing elements

```
>>> numbers[0]   #Returns first element
>>> numbers[-1]  #Returns last element
>>> numbers[9-8] #Index can be any expression resulting in integer
>>> numbers[1:3] #Slice: returns value at index 1 and 2
>>> numbers[:4]  #Slice: returns elements from 0 to 3
>>> numbers[3:]  #Slice: returns elements from 3 to last element
>>> numbers[:]   #Slice: returns all elements
```

- Lists are mutable: we can change their elements
- The function len returns the length of a list, which is equal to the number of its elements

List

- The “+” operator concatenates list and “*” operator repeats a list a given number of times.
- List Methods: Many in-built methods are available to work on lists.
  - **append, extend, pop, reverse, sort ……**
- The “pop” method will default pop the last element (LIFO), else can pop by passing the index
- Use “del” to delete an element from a list.

Tuple

- Tuples are similar to lists, but immutable.
- Creating tuples
  - rec = (“Ricky”, “IKP”, 1234)
  - point = x, y, z # parentheses optional
  - empty = () # empty tuple
- Tuple assignment: useful to assign multiple variables in one line
  - x, y, z = point # unpack
  - (a, b) = (b, a) # swap values
- Tuples can be used to return multiple values from a function.

Dictionary

- Dictionaries are hash tables or associative arrays.
- They map keys, which can be any immutable type, to values, which can be any type.
- Example:

```
>>> eng2sp = {}
>>> eng2sp["one"] = “uno”
>>> eng2sp["two"] = “dos”
>>> print(eng2sp)
{"two": "dos", "one": "uno"}
```

- Dictionaries are designed for very fast access using complex algorithms
- Dictionaries are mutable.

Dictionary

- As mentioned, the keys can be any immutable type. This allows even a tuple to be a key.

```
>>> matrix = {(0, 3): 1, (2, 1): 2, (4, 3): 3}
```

- Useful Functions:
  - dct.keys() #return a list of keys
  - dct.values() #return a list of values
  - dct.items() #return a list of key-value pairs
  - dct.has_key() #check for key existence in dictionary
  - dct.get('key', 1) #here if 'key' does not exist, then 1 will be returned

- Since dictionaries are mutable, so be aware of "aliasing". Use the copy() method to create a copy of original.
- Use del to delete elements in dictionary.

Sets

- "set" is a container that stores only unique elements.

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set(basket) # create a set without duplicates
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])
>>> 'orange' in fruit # fast membership testing
True
>>> 'crabgrass' in fruit
False
>>> # Demonstrate set operations on unique letters from two words ...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a # unique letters in a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b # letters in a but not in b
set(['r', 'd', 'b'])
>>> a | b # letters in either a or b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b # letters in both a and b
set(['a', 'c'])
>>> a ^ b # letters in a or b but not both
set(['r', 'd', 'b', 'm', 'z', 'l'])
```

Data Type Conversion

Function	Description
int(x [,base])	Converts x to an integer. base specifies the base if x is a string.
long(x [,base])	Converts x to a long integer. base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.
oct(x)	Converts an integer to an octal string.

Assignment operator

- Multiple assignments

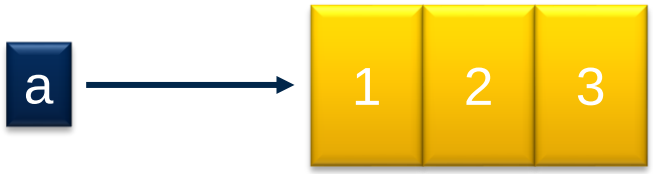
a = b = c = 1	An integer object is created with the value 1, and all three variables are assigned to the same memory location
a, b, c = 1, 2, "john"	two integer objects with values 1 and 2 are assigned to variables a and b, and one string object with the value "john" is assigned to the variable c

Reference Semantics

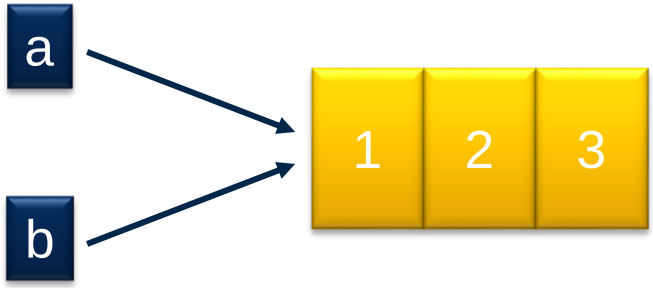
- There is difference in how python does assignment.
- Assignment manipulates reference.
  - x = y #makes x **reference** the object y references
  - x = y #**does not make a copy** of y
- Demo

Changing a Shared List

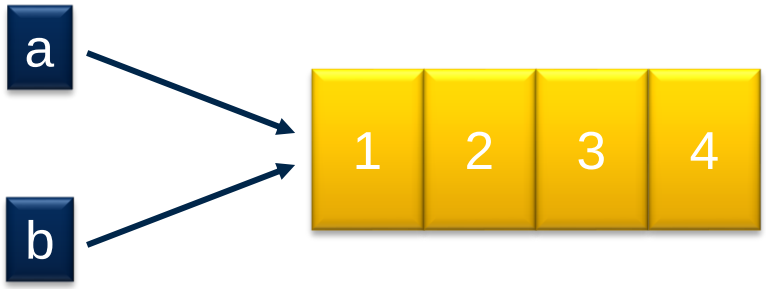
a = [1, 2, 3]



b = a



a.append(4)

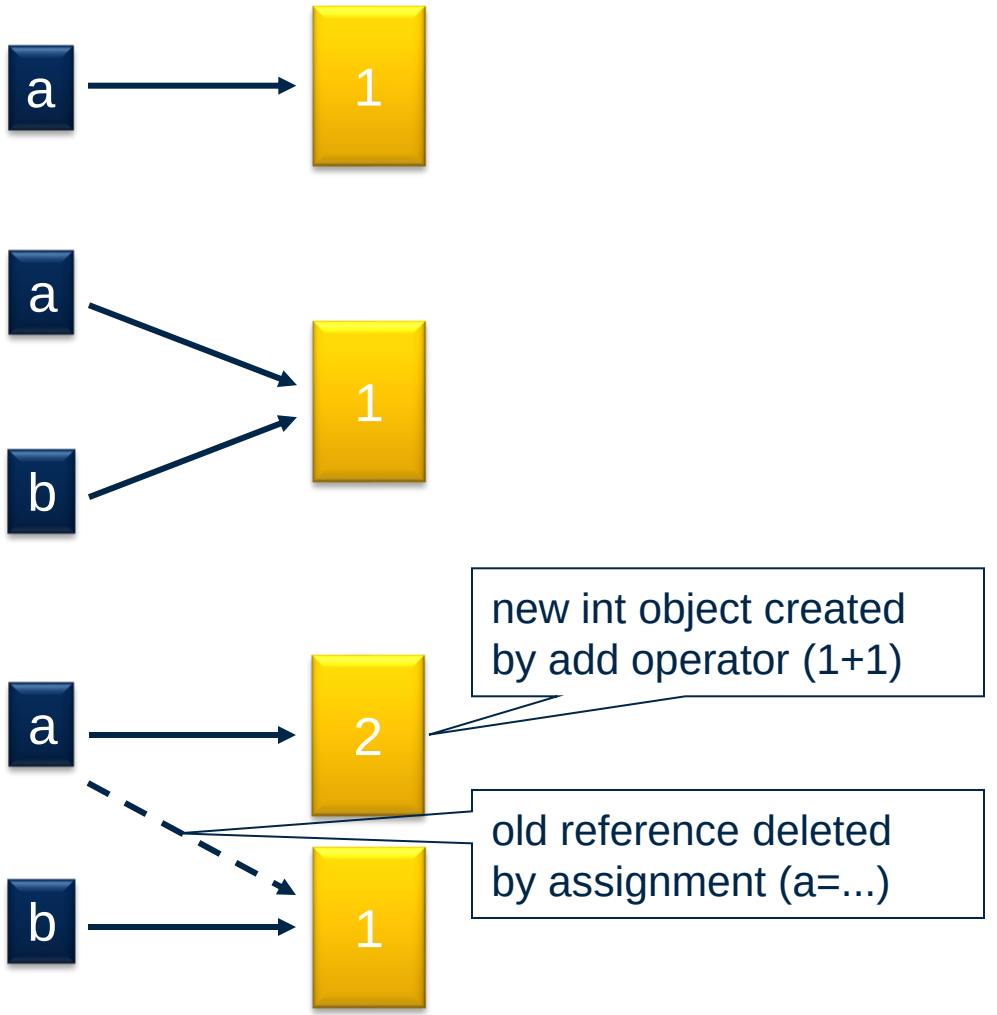


Changing an integer

a = 1

b = a

a = a+1



Control Structures

if.. elif.. else

Loops

if.. elif.. else..

- If… elif… else…

```
if x < y:
  STATEMENTS_A
elif x > y:
  STATEMENTS_B
else:
  STATEMENTS_C
```

- Ternary operator supported but generally avoided for clarity

```
i=1 if 10>20 else 2
```

- Single line if statement

```
if ( var == 100 ) : print "Value of expression is 100“
```

Loops

<table>
<tr><td>while loop</td><td colspan="2">while expression:
  statement(s)</td></tr>
<tr><td></td><td colspan="2">#Else will be executed if expression is false
while expression:
  statement(s)
else:
  statement(s)</td></tr>
<tr><td>for loop</td><td colspan="2">for iterating_var in sequence:
  statements(s)</td></tr>
<tr><td></td><td colspan="2">for iterating_var in sequence:
  statements(s)
else:
  statement(s)</td></tr>
<tr><td></td><td colspan="2">Useful functions for sequencing:</td></tr>
<tr><td></td><td>• range() / xrange()
• enumerate()
• zip()</td><td>• reversed()
• sorted()
• dct.iteritems()</td></tr>
</table>

Loops…cont.

Control Statement	Description
break	Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
pass	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

Introduction to Functions

- The syntax for function definition is

```
def NAME( PARAMETERS ):
    """Docstring"""
     STATEMENTS
  [return]
```

- A function must be defined before its first use
- The return statement is used to return a value from function.
- If a function does not have return statement , it is considered as a Procedure
- If a function has to return multiple values , tuples are preferred
- Sample function call

```
name = my_func(arg1, arg2, arg='Default')
```