



Python Syllabus

Basics

- ✓ 1. PYTHON SETUP
- ✓ 2. TUPLES Ch 2
- ✓ 3. TOKENS
- ✓ 4. DICTIONARIES Ch 4
- ✓ 5. FUNCTIONS Ch 1-11
- ✓ 6. DATA TYPES Ch 3
- 7. MODULES
- 8. OPERATIONS
- 9. PYTHON DEBUGGER
- 10. BRANCHING
- 11. LOOPS
- 12. SORTING
- 13. SEARCHING
- 14. LISTS Ch 2

OOPS

- 1. CLASSES
- 2. OBJECTS
- 3. METHODS
- 4. OVERRIDING
- 5. INHERITANCE
- 6. DATA HIDING
- 7. EXCEPTION
- 8. HANDLING

Scripting

- 1. REGRESSION
- 2. EXPRESSION
- 3. STRINGS
- 4. FILE
- 5. COMMAND LINE
- 6. ARGS

Python : Beginning Python From Novice to Professional

13/11/2021

Operators: +, -, *, /, //, %, **

// - This discards the Fractional part (puntos decimales)

```
>> 5/3  
1.666  
>> 5//3  
1.0
```

% - remainder of the division

```
>> 10%3  
1.0
```



** - exponential (power of)

```
>> 2 ** 3  
8  
>> -3 ** 2  
-9  
>> (-3)** 2  
9
```

Hexadecimal Octal and Binary

```
>> 0xAF      - Hexadecimal  
175  
>> 0o10      - Octal  
8  
>> 0b1011010010 - Binary  
722
```

convert Decimals to
hex()
oct()
bin()

Getting Input from the User: `input()` gives us a piece of text, or string.

```
>> input("x: ")
x: 34
>> input("y: ")
y: 42
>> print(int(x) * int(y))
1428
```

To convert it to a number we need the `>> int()` command

Some functions:

- power function:

$2^{**}3 = \text{pow}(2, 3)$; both do the same thing

Example:

```
>> 10 + pow(2, 8*5) / 3.0
10932.666
```

Other expressions are:

- `abs(-10)`

10 - absolute value of a number

- `round(2/3)`

1.0 - rounds floating-point number to the nearest integer.

Modules Math:

First write

```
>> import math
```

This will allow you to use other expressions like `math.floor()`,
`math.ceil()`, `math.sqrt()`
rounds up

If you are sure that you won't import more than one function with a given name (from different modules), you might not want to write the module name each time you call the function. Then you can use a variant of the `import` command.

```
>>> from math import sqrt
>>> sqrt(9)
3.0
```

After using the `from module import` function, you can use the function without its module prefix.

One may use variables to refer to functions (and most other things in Python)

```
>> foo = math.sqrt
```

then use

```
>> foo(4)
```

or

+ For complex numbers we need to import "cmath" [Another module]

```
>> import cmath
```

```
>> cmath.sqrt(-1)
```

Complex math

j

>> (1 + 3j) * (9 + 4j) Python has complex number support for sum.

(-3 + 31j)

Modules

pandas : graphs and lists

sys : Systems

tweepy

Chapter 2: Lists and Tuples (Sequences)

One can change a list, but not a tuple

Name and Age:

```
>>> edward = ['Edward Gumby', 42]
```

```
>>> john = ['John Smith', 50]
```

Sequence can contain other sequences

```
>>> database = [edward, john]
```

```
>>> database
```

```
[['Edward Gumby', 42], ['John Smith', 50]]
```

Indexing:

- All elements in a sequence are numbered - from zero and upward. You can access them individually with a number, like this:

```
>>> greeting = 'Hello'
```

```
>>> greeting[0]  
'H'
```

```
>>> greeting[-1]  
'o'
```

- If a function call returns a sequence, you can index it directly

I want the fourth number of the year inputed

```
>>> fourth = input('Year: ') [3]
```

Year: 2005

```
>>> fourth  
15
```

It When fourth called it will always get the last number
One can also put [-1] instead

Slicing

- Just as you use indexing to access individual elements, you can use slicing to access ranges of elements. You do this by using two indices, separated by a colon.

```
>>> tag = '<a href="http://www.python.org"> Python web site </a>'
```

```
>>> tag[9:30]  
'http://www.python.org'
```

```
>>> tag[32:-4]  
'Python web site'
```

- The first index is the number of the first element you want to include. However, the last index is number of the first element after you slice. Consider the following:

```
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
>>> numbers[3:6] [4, 5, 6]
```

```
>>> numbers[0:1] [1]
```

Shortcut: Access allways the last three elements of any size list

```
>>> numbers[7:10] # If known size
```

```
[9, 9, 10]
```

Wrong:

```
>>> numbers[-3:-1]
```

```
[8, 9]
```

correct

```
>>> numbers[-3:]
```

```
[8, 9, 10]
```

```
>>> numbers[-3:0]
```

```
[]
```

Also one can do:

```
>>> numbers[:3] [1, 2, 3]
```

and

```
>>> numbers[:] [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Longer Steps: One can put another parameter to put the steps that will it move from one element to the other.

```
>>> number = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
>>> number[0:10:1] one element at a time
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
>>> number[0:10:2]
```

```
[1, 3, 5, 7, 9]
```

```
>>> number[3:6:3] >>> number[3:7:3]
```

```
[4]
```

```
[4, 7]
```

Other examples:

```
>>> numbers[::4]
```

```
[1, 5, 9]
```

```
>>> numbers[:::-2]
```

```
[10, 8, 6, 4, 2]
```

```
>>> numbers[8:3:-1]
```

```
[9, 8, 7, 6, 5]
```

```
>>> numbers[5::-2]
```

```
[6, 4, 2]
```

```
>>> numbers[10:0:-2]
```

```
[10, 8, 6, 4, 2]
```

```
>>> numbers[5::-2]
```

```
[10, 8]
```

```
>>> numbers[0:10:-2]
```

```
[]
```

Adding and multiplying Sequences

```
>>> [1, 2, 3] + [4, 5, 6]
```

```
[1, 2, 3, 4, 5, 6]
```

```
>>> 'Hello, ' + 'world!'
```

```
'Hello, world!'
```

```
>>> 'python' * 5
```

```
'pythonpythonpythonpythonpython'
```

```
>>> [42] * 5
```

```
[42, 42, 42, 42, 42]
```

Cannot add strings and lists

- None , Empty Lists , and Initialization -

'None' is actually a value of nothing

one can make a sequence of nothing

```
>>> sequence = [None]*10
```

```
[None, ..., None]
```

- Membership - (Check if something is in a sequence)

One can use Boolean operators to see if a value can be found in sequence. Boolean checks if something is true or false.

Examples:

```
>>> permissions = 'rw'
```

```
>>> 'w' in permissions
```

True

```
>>> 'x' in permissions
```

False

```
>>> users = ['m1h', 'foo', 'bar']
```

```
>>> input('Enter your user name: ') in users
```

Enter your user name: m1h

True

* See example of Checking if a username and a Pin code is in a sequence in Indexing Problems.

- Length , Minimum , and Maximum -

Pretty Straight Forward

```
>>> numbers = [100, 34, 678]
```

```
>>> len(numbers)
```

3

```
>>> max(numbers)
```

678

```
>>> min(numbers)
```

34

Lists: Python Workhorse (Everything you can do with lists)

Commands

List

```
>>> list('Hello')
['H', 'e', 'l', 'l', 'o']
```

Join ??

```
''.join(somelist)
```

Item Assignment

```
>>> x = [1, 1, 1]
>>> x[1] = 2
[1, 2, 1]
```

Delete Element

```
>>> name = ['Alice', 'Beth', 'Juan']
>>> del names[2]
['Alice', 'Beth']
```

Slicing

```
>>> name = list('Pearl')
['P', 'e', 'r', 'l']
>>> name[1:] = list('ython')
>>> name
['P', 'y', 't', 'h', 'o', 'n']
```

Inserting Slice

```
>>> numbers = [1, 5]
>>> numbers[1:1] = [2, 3, 4]
>>> numbers
[1, 2, 3, 4, 5]
```

Deleting Slice

```
>>> numbers
[1, 2, 3, 4, 5]
>>> numbers[1:4] = []
[1, 5]
```

Count

```
>>> ['to', 'be', 'or', 'not', 'to', 'be'].count('to')
```

2

```
>>> x = [1, 2], 1, 1, [2, 1, [1, 2]]]
```

```
>>> x.count(1)
```

2

```
>>> x.count([1, 2])
```

1

Index

```
>>> Knights = ['We', 'are', 'the', 'Knights', 'who', 'say', 'ni']
```

```
>>> Knights.index('who')
```

4

Insert

```
>>> numbers = [1, 2, 3, 5, 6, 7]
>>> numbers.insert(3, 'four')
>>> numbers
[1, 2, 3, 'four', 5, 6, 7]
```

Methods

object.method(arguments)

append

```
>>> lst = [1, 2, 3]
>>> lst.append(4)
[1, 2, 3, 4]
```

clear

```
>>> lst = [1, 2, 3]
>>> lst.clear()
[]
```

copy

```
>>> a = [1, 2, 3]
>>> b = a.copy()
>>> b[1] = 4
[1, 4, 3]
```

same lists just
different lists

extend

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a.extend(b)
>>> a
[1, 2, 3, 4, 5, 6]
```

careful

```
>>> a + b
[1, 2, 3, 4, 5, 6]
>>> a
[1, 2, 3]
```

pop (removes last element by default)

```
>>> x = [1, 2, 3]
>>> x.pop()
3
>>> x.pop()
1
```

x

stack can
be created

with LIFO
x.append(x.pop())

remove

```
>>> x = ['to', 'be', 'or', 'be']
>>> x.remove('be')
['to', 'or', 'be']
```

* FIFO

x.insert(x.pop()) or
x.append(x.pop())



reverse

```
>>x = [1, 2, 3]
>>x. reverse()
>>x
[3, 2, 1]
```

sort and sorted function
 $\ggg x = [4, 6, 2, 1, 7, 9]$ }
 $\ggg x. sort()$
 $[1, 2, 4, 6, 7, 9]$
 will always return a list

$\ggg y = sorted(x)$

$\ggg x$

$[4, 6, 2, 1, 7, 9]$

$\ggg y$

$[1, 2, 4, 6, 7, 9]$

sort examples

```
>>x = ['aardvark', 'abalone', 'acme', 'add', 'anare']
>>x.sort(key=len) # Will sort by the length of the word
>>x = [4, 6, 2, 1, 7, 9]
>>x.sort(reverse=True)
>>x
[9, 7, 6, 4, 2, 1]
```

Tuples: Immutable sequences

written with parenthesis

```
>> 1, 2, 3
(1, 2, 3)
>> (1, 2, 3)
(1, 2, 3)
```

Empty tuple

$\ggg ()$

$()$

Tuples with one value

```
>> 42,
(42,)
>> (42,)
(42,)
```

Some examples

$\ggg 3 * (10 + 2)$, but $\ggg 3 * (40 + 2,)$
 126 $(42, 42, 42)$

Tuple function

```
>>tuple([1, 2, 3])
(1, 2, 3)
>>tuple('abc')
('a', 'b', 'c')
```

One can only create them and access their elements

```
>> x = 1, 2, 3
>> x[1]
2
>> x[0:2]
(1, 2)
```

Chapter 2 : Summary

New Functions in This Chapter

Function	Description
<code>len(seq)</code>	Returns the length of a sequence
<code>list(seq)</code>	Converts a sequence to a list
<code>max(args)</code>	Returns the maximum of a sequence or set of arguments
<code>min(args)</code>	Returns the minimum of a sequence or set of arguments
<code>reversed(seq)</code>	Lets you iterate over a sequence in reverse
<code>sorted(seq)</code>	Returns a sorted list of the elements of seq
<code>tuple(seq)</code>	Converts a sequence to a tuple

Chapter 3: Working with Strings

All standard sequences operations (indexing, slicing, multiplication, membership, length, minimum and maximum) work with strings.

String replacement

```
>>> format = "Hello, %s. %s enough for ya?"  
>>> values = ('world', 'Hot') # Tuple  
>>> format % values  
'Hello, world. Hot enough for ya?'
```

The %s format string are called conversion specifiers. They mark places where the values are to be inserted. The s means that the values should be formatted as if they were strings; if they aren't, they'll be converted with str.

One can also use:

% .3f ~ value as a floating-point number with three decimals

String Methods (For more see Appendix B)

One can use the `string` module to use these string methods that will be available

BUT STRING ISN'T DEAD

Even though string methods have completely upstaged the `string` module, the module still includes a few constants and functions that aren't available as string methods. The following are some useful constants available from `string`:

- `string.digits`: A string containing the digits 0–9
- `string.ascii_letters`: A string containing all ASCII letters (uppercase and lowercase)
- `string.ascii_lowercase`: A string containing all lowercase ASCII letters
- `string.printable`: A string containing all printable ASCII characters
- `string.punctuation`: A string containing all ASCII punctuation characters
- `string.ascii_uppercase`: A string containing all uppercase ASCII letters

Despite explicitly dealing with ASCII characters, the values are actually (unencoded) Unicode strings.

String Methods that can be used without the module -

center (centers the string by padding it on either side)

```
>>> "Juan is learning".center(39)
'Juan is learning'
```

>>> "Juan is learning".center(39, '^')
'^ Juan is learning ^'

find (Finds a substring within a larger string. Returns the leftmost index where the substring is found. If not found, -1 is returned.)

```
>>> title = "Monty Python's Flying Circus"
>>> title.find('Monty')          >>> title.find('Zinguss')
0                                -1
>>> title.find('Python')        6
>>> title.find('Flying')       15
```

One can also supply a starting point for a search and an ending point.

```
>>> subject = '$$$ Get rich now!!! $$$'
>>> subject.find('$$$')          >>> subject.find('!!!')
0                                16
>>> subject.find('$$$', 1)       >>> subject.find('!!!', 0, 16) # start and end
20                               -1
```

