Python in a Couple of Hours

A Python Programming Tutorial for Short Attention Spans.

1.Why Python?

Python is an object-oriented, high-level interpreted language originally developed in the early '90s by Guido van Rossum. The goal of Python was to be a language that could be used to teach the most advanced concepts of programming to non-programmers.

Python is:

- Compact (33 keywords) - Simple to remember.
- Easy to learn - Easy to remember. Less time looking things up (looking at you Java).
- Very High-level – Very English-like. “programming at the speed of thought”
- Terse / non-verbose – Python programs are typically ¼ to 1/10th the size of C++/Java.
- Object Oriented – Allows OOP, doesn't force it. Very purely OOP under the hood.
- Cross Platform – Windows, LINUX, UNIX, Mac, Android, embedded systems, etc.
- Powerful – Can do most things other languages can. GUI, 2d/3d graphics, sound, etc.
- Community Owned / Developed – www.python.org
- Large user-base – Google, ILM, Disney, NASA, EA games, 2K games, Redhat, Spotify, Amazon, Instagram, Facebook, Netflix, Reddit, etc.
- Free – Free as in Beer, Free as in Speech.

2.Downloading and Installing.

Python is freely available from: http://www.python.org. Follow the instructions to install.

IDLE, a simple IDE, is part of the default install.

3.Modes

Python has two modes:

- **Immediate (or interactive) mode** – gives a python shell command prompt that can be interacted with. Useful for testing things or trying out simple ideas.
- **Script mode** – runs a pre-written python script to completion. Used for running and testing more complex programs.

4. Python Syntax

4.1 Properties

- Strongly typed (types are enforced)
- Dynamically/implicitly typed (you don't have to declare variables)
- Case sensitive (i.e. xvalue and xValue are two different variables)
- Fully object-oriented (i.e. everything is an object, there are no native types).

4.2 Indentation

Blocks are specified by indentation rather than statement termination characters.

Indent in to begin a block, indent out to end one.

All statements that expect an indentation level end in a colon (:).

4.3 Syntax

4.3.1 Comments

Comments start with the pound (#) sign and continue to the end of the line.

Here are some comments:

```
# This is a comment
print("hello world") # this is a comment
```

There is no way to block comment.

4.3.2 Strings

4.3.2.1 String Basics

In python, a string is a special type of sequence object that holds an immutable sequence of characters.

Strings can be created using double quotes, single quotes, and triple quotes.

- `"This is a string"`
- `'This is a string'`

-
  ```
   ''' This is
  a string '''
  ```

Strings are objects and have methods:

```
>>> 'The ugly dog ran home.'.upper()
'THE UGLY DOG RAN HOME.'
>>> 'The ugly dog ran home.'.find('dog')
9
>>> 'The ugly dog ran home.'.find('cat')
-1
>>> 'The ugly dog ran home.'.replace('dog','cat')
'The ugly cat ran home.'
```

4.3.2.2 String Operators

Concatenation (+):

```
>>>"this" + "that"
'thisthat'
```

Repitition (*):

```
>>>"this"*4
'thisthisthisthis'
>>>4*"this"*2
'thisthisthisthisthisthisthisthis'
```

There is also a string formatting operator: "%".

```
>>> name = 'Paul'
>>> 'Welcome to the class %s.' % name
'Welcome to the class Paul.'
```

You can use the following formatting characters with %:

- d,i Signed integer decimal.
- u Unsigned decimal.
- o Unsigned octal.
- x, Unsigned hexidecimal (lowercase).
- X Unsigned hexidecimal (uppercase).
- e Floating point exponential format (lowercase).
- E Floating point exponential format (uppercase).
- f,F Floating point decimal format.
- c Single character (accepts integer or single character string).
- s String (converts any python object using str()). (4)
- % No argument is converted, results in a "%" character in the result.

4.3.3 Numeric Types

4.3.3.1 Numeric Basics

The two most common Python numeric object types are integer numbers and floating-point numbers.

- Integers in python aren't limited in their precision. There are two types: integers, longs.
- Floating point numbers are based on the C language double type. (The standard module 'decimal' allows arbitrary precision floating point numbers.)

4.3.3.2 Numeric Operators

Here are some basic python numeric operators:

Unary Operators: +,-
Binary operators: +,-,*,/,%, **
Functions: abs( ), int( ), float( ), long( ), pow(x,y)
Grouping: ( )

4.3.4 Variables

4.3.4.1 Simple Assignments

Variables are created in python with the assignment operator as follows:

```
>>>x=10
>>>name="Paul"
>>>value=3.1415
```

In python what the assignment operator is really doing is binding a name to an object. Since python is dynamically typed, a name may be rebound at any time regardless of the type to which it is being bound. Thus the following is perfectly legal:

```
>>>x=10
>>>x="hello"
>>>x=3.1415
```

4.3.4.2 Modify and Assign

Python also allows modify-and-assign numeric operators +=, -=, *=, /=, %=, **=.

```
>>>x=10
>>>x+=5
>>>x
15
```

The += and *= operators also work on string objects.

4.3.4.2 Multi-Assignment

Python also allows multiple assignments on one line as follows:

```
>>> x,y = 10 , 5
```

is the same as saying:

```
>>> x=10
>>> y=5
```

4.3.5 Printing and Inputting

Textual output is realized using the print statements as follows:

```
>>> print("hello world")
hello world
```

Multiple items can printed by using commas between items:

```
>>> print("Ten hours is",10*60*60,"seconds")
Ten hours is 36000 seconds
```

A comma at the end causes additional items to be printed on the same line.

Print statements can also include escape sequences:

- \\ Backslash (\)
- \' Single quote (')
- \" Double quote (")
- \a ASCII Bell (BEL)
- \b ASCII Backspace (BS)
- \f ASCII Formfeed (FF)
- \n ASCII Linefeed (LF)
- \r ASCII Carriage Return (CR)
- \t ASCII Horizontal Tab (TAB)
- \v ASCII Vertical Tab (VT)
- \ooo Character with octal value ooo
- \xhh Character with hex value hh

Getting input from the keyboard is done using the raw_input() function.

```
str = input("Type in a String: ")
```

Note that input( ) always returns a string.

4.3.6 Comparisons

Python allows the use of special comparison operators that produce a True or False result. True and False are special names in Python.

The comparison operators are:

<	less than

<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal
!=	not equal
<>	another way to say not equal

There are several other logical operations that can be performed on comparisons:

( ) - parenthesis are used for grouping
and, **or**, and **not** are also used for combining comparisons.

4.3.7 Decisions

Python uses the if statement for decisions:

Simple if:

```
if age<21:
        print("No beer")
```

if else:

```
if age<21:
        print("No beer")
else:
        print("much beer 4u")
```

if elif else:

```
if age<21:
        print("No beer")
elif age>100:
        print("beer – you really deserve it!")
elif age>70:
        print("geritol beer")
else:
        print("much beer 4u")
```

Note that the indented code is the if statement body. Any number of lines may be indented and indented lines may include other nested decisions.

4.3.8 Loops

Python has two types of loop, the while loop and the for loop.

The while loop looks like this:

```
a = 0
while a < 10:
        a = a + 1
        print(a)
```

The for loop looks like this:

```
for number in [10,20,30,40]:
        print(number/10**2)
```

Note that the for loop works through each item of a sequence.

4.3.9 Defining Functions

Functions in Python are defined using the following syntax:

```
def hello():
    print("Hello")

def area(width,height):
    return width*height

def volume(width,height,depth):
    return width*height*depth

def distance(x1,y1,x2,y2):
    return ((x2-x1)**2+(y2-y1)**2)**0.5

def factorial(n):
    if n <= 1:
        return 1
    return n*factorial(n-1)
```

A few notes on variables and functions:

- Functions must be defined before they can be used.
- Variable assigned within functions are local to that function even is they share a name with a variable outside the function.
- Functions can access data from variables defined from the outter namespace, but cannot (by default) modify those variables' contents.
- Use the global statement to allow a function to change the value of a variable outside its namespace.

4.3.10 Importing and Using Library Functions

Libraries are accessed by using the import statement.

```
import random
random.randint(0,10)
random.uniform(-1.5,1.5)
```

or the from statement

```
from random import randint
randint(0,-10)
```

4.3.11 Sequences

Python supports six sequence types: strings, Unicode strings, lists, tuples, buffers, and xrange objects.

We've already looked at strings previously and we're only going to look in detail at lists and tuples in this section:

String: an immutable sequence of characters. S1 = "hello"

List: a mutable sequence of arbitrary items. L1 = [1,2,3,4,5]

Tuple: an immutable sequence of arbitrary items. T1 = (0, 2, 4, 6)

All sequences support the following methods / functions:

x in s	True if an item of s is equal to x, else False (1)
x not in s	False if an item of s is equal to x, else True (1)
s + t	the concatenation of s and t (6)
s * n , **n * s**	n shallow copies of s concatenated (2)
s[i]	i'th item of s, origin 0 (3)
s[i:j]	slice of s from i to j (3), (4)
s[i:j:k]	slice of s from i to j with step k (3), (5)
len(s)	length of s
min(s)	smallest item of s
max(s)	largest item of s

Note that mutable sequences can have items changed in place (s[2] = "hello").

Mutable Sequences:

Mutable Sequences have additional operators / methods that apply:

s[i] = x	item i of s is replaced by x
s[i:j] = t	slice of s from i to j is replaced by t
del s[i:j]	same as s[i:j] = []
s[i:j:k] = t	the elements of s[i:j:k] are replaced by those of t
del s[i:j:k]	removes the elements of s[i:j:k] from the list
s.append(x)	same as s[len(s):len(s)] = [x]
s.extend(x)	same as s[len(s):len(s)] = x
s.count(x)	return number of i's for which s[i] == x
s.index(x[, i[, j]])	return smallest k such that s[k] == x and i <= k < j
s.insert(i, x)	same as s[i:i] = [x]
s.pop([i])	same as x = s[i]; del s[i]; return x
s.remove(x)	same as del s[s.index(x)]
s.reverse()	reverses the items of s in place
s.sort([cmp[, key[, reverse]]])	sort the items of s in place

4.3.12 More with the For Loop

The for loop processes any sequence by going through each item in the sequence.

The range function automatically creates a sequence of items and can be used to make the for statement work like a traditional for loop.

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 11)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(0, 30, 5)
[0, 5, 10, 15, 20, 25]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(0, -10, -1)
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> range(0)
[]
>>> range(1, 0)
[]
```

4.3.13 Dictionaries

Dictionaries are mutable objects that hold key/value pairs.

Dictionaries are created by placing a comma-separated list of key: value pairs within braces, for example: {'jack': 4098, 'sjoerd': 4127} or {4098: 'jack', 4127: 'sjoerd'}.

The following operations are defined on dictionaries (where a and b are mappings, k is a key, and v and x are arbitrary objects):

- len(a) the number of items in a
- a[k] the item of a with key k
- a[k] = v set a[k] to v
- del a[k] remove a[k] from a
- a.clear() remove all items from a
- a.copy() a (shallow) copy of a
- a.has_key(k) True if a has a key k, else False
- k in a Equivalent to a.has_key(k)
- k not in a Equivalent to not a.has_key(k)
- a.items() a copy of a's list of (key, value) pairs
- a.keys() a copy of a's list of keys
- a.update([b]) updates (and overwrites) key/value pairs from b
- a.fromkeys(seq[, value]) Creates a new dictionary with keys from seq and values set to value
- a.values() a copy of a's list of values
- a.get(k[, x]) a[k] if k in a, else x
- a.setdefault(k[, x]) a[k] if k in a, else x (also setting it)
- a.pop(k[, x]) a[k] if k in a, else x (and remove k)
- a.popitem() remove and return an arbitrary (key, value) pair
- a.iteritems() return an iterator over (key, value) pairs
- a.iterkeys() return an iterator over the mapping's keys
- a.itervalues() return an iterator over the mapping's values

4.3.14 Classes and Object Instances

Python classes are created with the class statement. Methods are simply functions defined within the class definition and attributes are created in the __init__ function by using the self reference. The self reference is a special parameter that refers to the instance itself.

```
Class Coin(Object):
        def __init__(self):
                self.side="heads"
        def turnover(self):
                if self.side=="heads":
                        self.side="tails"
                else:
                        self.side="heads"
        def flip(self):
                import random
                self.side=["heads","tails"][random.randint(0,1)]
```

Object instances are created with the defined class as follows:

```
penny = Coin( )
```

Methods/Attributes are accessed using the object instance and the name of the appropriate attributes / methods.

```
penny.flip()
print(penny.side)
```

__init__ is a special method that acts as the object's constructor.

The constructor can have parameters passed to it by adding additional parameters to the __init__ function.

There are other special methods such as: __del__(self) and __str__(self)

4.3.15 File I/O

File I/O in python uses several built-in functions.

Opening Files:

```
fh1 = open("data.txt","w") # open file for writing
fh2 = open("otherdata.txt","r") # open file for reading
```

Writing to Files:

```
fh1.write("Hello")
fh1.write("World\n")
fh1.write("Second Line\n")
fh1.write("Third Line\n")
```

* Note that the write method only accepts strings.

```
fh1.writelines(["Paul","Jim","Jason","Duane"])
```

*Note that the writelines method doesn't add newlines or any other type of separator.

Reading From Files:

```
ch = fh2.read(1)   # read a single character
chs = fh2.read(3)  # read three characters
all = fh2.read()   # read all the remaining characters
line = fh2.readline() # read an entire line (includes newline)
lines = fh2.readlines() # read all the remaining lines returns them as a list.

for line in fh2:      # iterate over the lines of the file.
       print(line)
```

Closing Files:

```
fh2.close()   # close the file

fh2.closed    # use this property to see if a file is closed
```

Controlling the File Position:

```
fh2.seek(0) # rewind to the beginning

fh2.seek(10) # jump to character 10 in the file

fh2.seek(10,1) # jump ahead 10 from current position
fh2.seek(-10,2) # jump to the position 10 before the end of the file
```

Storing More Complex Data:

For storing more complex data types see the cPickle modules and the shelve module.

pickle and cPickle:

```
cPickle.dump(object,file) #pickle and store the object in the opened file.
st=cPickle.dumps(object) # pickle the object and return it as a string.
object=cPickle.load(file)  # loads and unpickles the data in the file.
object=cPickle.loads(string) # unpickles the data in the string.
```

Shelve:

```
        items=shelve.open("test.dat")
        items["high_score_name"]="Paul"
        items["high_score_value"]=1000
        items.sync()   # update the file.
        items.close()

        items=shelve.open("test.dat")
        print(items["high_score_name"])  # prints Paul
        print(items["high_score_value"]) # prints 1000
        items.close()
```

4.3.16 Error Handling

Unhandled exceptions cause python to print the error message and halt the program. The following code can raise an exception:

```
n1 = input("Enter a number:")
n2 = input("Enter another number:")
result=int(n1)*int(n2)
print(n1,"multiplied by",n2,"equals",result)
```

To catch errors in code we use the following construct:

```
try:
        n1 = input("Enter a number:")
        n2 = input("Enter another number:")
        result=int(n1)*int(n2)
        print(n1,"multiplied by",n2,"equals",result)
except:
        print("You are evil")
```

We can also specify what type of error we'd like to catch:

```
try:
        n1 = input("Enter a number:")
        n2 = input("Enter another number:")
        result=int(n1)*int(n2)
        print(n1,"multiplied by",n2,"equals",result)
except(Exception):
        print("You are evil")
```

Or we can specify multiple exception types:

```
try:
        n1 = input("Enter a number:")
        n2 = input("Enter another number:")
        result=int(n1)*int(n2)
        print(n1,"multiplied by",n2,"equals",result)
except(TypeError,ValueError):
        print("You are evil")
```

or:

```
try:
        n1 = input("Enter a number:")
        n2 = input("Enter another number:")
        result=int(n1)*int(n2)
        print(n1,"multiplied by",n2,"equals",result)
except(TypeError):
        print("wrong type moron!")
except(ValueError):
        print("You are evil")
```

Or you can add an else to the end. The else clause will run if there are no exceptions:

```
try:
        n1 = input("Enter a number:")
        n2 = input("Enter another number:")
        result=int(n1)*int(n2)
        print(n1,"multiplied by",n2,"equals",result)
except(TypeError,ValueError):
        print("You are evil")
else:
        print("no errors")
```

Finally, python can give you a description of the error:

```
try:
        n1 = input("Enter a number:")
        n2 = input("Enter another number:")
        result=int(n1)*int(n2)
        print(n1,"multiplied by",n2,"equals",result)
except(Exception), description:
        print("You are evil and caused this disaster:", description)
```

5. Additional Topics

5.1 2D Graphics

For simple 2d graphics and games I'd recommend the **pygame** module (which is available from www.pygame.org).

Additionally, I've created **pscreen** which is a simplified pygame wrapper that makes accessing the most common graphics and I/O functions much easier to use. Pscreen documentation and the module download are available from my web page at: http://tech.yostengineering.com/pscreen

For 2D graphics and image manipulation the Python Imaging Library is of interest: http://www.pythonware.com/products/pil/

5.2 3D Graphics

There are many 3D graphics modules available:

pyopengl - http://pyopengl.sourceforge.net/
cgkit - http://cgkit.sourceforge.net/
pyogre - http://www.ogre3d.org/wiki/index.php/PyOgre
panda3d - http://panda3d.org/
vpython - http://www.vpython.org/
directpython – http://directpython.sourceforge.net/

For other modules visit:

http://www.vrplumber.com/py3d.py

5.3 GUI Toolkits

There are many GUI toolkit options available for building fully graphical GUI applications:

Tk - http://wiki.python.org/moin/TkInter
WxPython - http://www.wxpython.org/
pyQt - http://wiki.python.org/moin/PyQt