Basics of Python

by Kaustubh Vaghmare

(IUCAA, Pune)

E-mail: kaustubh[at]iucaa[dot]ernet[dot]in

Topics to be Covered

(Not in any specific order.)

- Basic I/O in Python
- Data Types in Python
- Programming Philosophy
- Under The Hood
- Conditionals
- Loops
- Function Basics

Assumptions!!!

You are not new to programming.

You are new to Python!

Python 2 or 3?

- Python's key strength lies in its libraries.
- These are not ready for Python 3 yet.
- But they soon(!) will be!

Keep track of progress & Migrate!

Our First Program!

In [3]:
```
a = 3
b = 5
c = a+b
d = a-b
q, r = a/b, a%b # Yes, this is allowed!

# Now, let's print!
print "Hello World!" # We just had to do this, did we not?
print "Sum, Difference = ", c, d
print "Quotient and Remainder = ", q, r
```

```
Hello World!
Sum, Difference =  8 -2
Quotient and Remainder =  0 3
```

What can we learn from this simple program?

Dynamic Typing

- Never declare variables and types in advance.
- Variables created when first assigned values.
- Variables don't exist if not assigned.

Commenting

Everything after # is a comment and is ignored.

"print" statement

Replaced by a print() function in Python 3.

Tuple unpacking assignments

```
a,b = 5,6
```

More complicated forms introduced in Python 3.

Other Things

- Behavior of / and % operators with integer types.
- No termination symbols at end of Python statements.
- Exception to the above...

  a = 3; b = 5

Under the Hood

- No explicit compiling/linking step. Just run... $ python First.py
- Internally, program translated into bytecode (.pyc files)
- The "translation + execution" happens line-by-line

Implications of ”line-by-line” style

- N lines will be executed before error on N+1th line haults program!
- An interactive shell.

[Interactive Shell Demo]

[Introduction to iPython]

The First Tour of the Data Types

- Numbers - Integers
- Numbers - Floats

(Exploration of math module)

- Strings

(Methods of Declaring Strings)

(Concept of Sequences)

(Concept of Slicing)

(Concept of Mutability)

(Introduction of Object.Method concepts)

Integers

In [4]:
```
8 ** 2 # Exponentiation
```

Out[4]:
```
64
```

In [6]:
```
23**100 # Auto-upgrade to "LONG INT" Notice the L!
```

Out[6]:
```
1488619150636303939379155658655975423198711965380136868657698820922243327853933135215239014327734680423347659217944731085952022529876001L
```

In [7]:
```
5 / 4, 5%4 # Quotient-Remainder Revisited.
```

Out[7]:
```
(1, 1)
```

Floats

In [8]:
```
5.0 * 2, 5*2.0 # Values upgraded to "higher data type".
```

Out[8]: (10.0, 10.0)

In [9]:
```
5**0.5 # Yes, it works! Square-root.
```

Out[9]: 2.23606797749979

In [10]:
```
5 / 4.0 # No longer a quotient.
```

Out[10]: 1.25

In [12]:
```
5 % 4.0 # Remainder, yes!!!
```

Out[12]: 1.0

Math Module

- A module can be thought of as a collection of related functions.
- To use a module,

  import ModuleName
- To use a function inside a module, simply say

  ModuleName.Function(inputs)

Let's see the math module in action!

In [13]:
```
import math
x = 45*math.pi/180.0
math.sin(x)
```

Out[13]:
```
0.7071067811865475
```

In [14]:
```
math.sin( math.radians(45) ) # nested functions
```

Out[14]:
```
0.7071067811865475
```

There are about 42 functions inside Math library! So, where can one get a quick reference of what these functions are, what they do and how to use them!?!?

In [15]:
```
print dir(math) # Prints all functions associated with Math modu
le.
```

```
['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin',
 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot'
, 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc']
```

In [16]:
```
help(math.hypot)
```

```
Help on built-in function hypot in module math:

hypot(...)
    hypot(x, y)

    Return the Euclidean distance, sqrt(x*x + y*y).
```

Strings

There are three methods of defining strings.

In [17]:
```
a = "John's Computer" # notice the '
```

In [18]:
```
b = 'John said, "This is my computer."' # notice the "
```

In [19]:
```
a_alt = 'John\'s Computer' # now you need the escape sequence \
```

In [1]:
```
b_alt = "John said, \"This is my computer.\"" # again escape seq
uence.
```

In [2]:
```
long_string = """Hello World!

I once said to people, "Learn Python!"

And then they said, "Organize a workshop!" """
```

In [4]:
```
long_string_traditional = 'Hello World! \n\nI once said to peopl
e, "Learn Python!" \
\n\nAnd then they said, "Organize a workshop!" '
```

- Can be used to dynamically build scripts, both Python-based and other "languages".
- Used for documenting functions/modules. (To come later!)

String Arithmetic

In [1]:
```
s1 = "Hello" ; s2 = "World!"
```

In [2]:
```
string_sum = s1 + s2
print string_sum
```

```
HelloWorld!
```

In [3]:
```
string_product = s1*3
print string_product
```

```
HelloHelloHello
```

In [4]:
```
print s1*3+s2
```

```
HelloHelloHelloWorld!
```

String is a sequence!

In [5]:
```
a = "Python rocks!"
```

In [6]:
```
a[0], a[1], a[2] # Positions begin from 0 onwards.
```

Out[6]:
```
('P', 'y', 't')
```

In [7]:
```
a[-1], a[-2], a[-3] # Negative indices - count backwards!
```

Out[7]:
```
('!', 's', 'k')
```

In [8]:
```
len(a) # Measures length of both sequence/unordered collections!
```

Out[8]:
```
13
```

Sequences can be sliced!

In [9]:
```
a[2:6] # elements with indices 2,3,4,5 but not 6
```

Out[9]:
```
'thon'
```

In [10]:
```
a[8:-2] # indices 8,9 ... upto 2nd last but not including it.
```

Out[10]:
```
'ock'
```

In [11]:
```
a[:5] # Missing first index, 0 assumed.
```

Out[11]:
```
'Pytho'
```

In [12]:
```
a[5:] # Missing last index, len(a) assumed.
```

Out[12]:
```
'n rocks!'
```

Crazier Slicing

In [14]:
```
a[1:6:2],a[1],a[3],a[5] # Indices 1, 3, 5
```

Out[14]:
```
('yhn', 'y', 'h', 'n')
```

In [15]:
```
a[::2] # beginning to end
```

Out[15]:
```
'Pto ok!'
```

In [16]:
```
a[::-1] # Reverse slicing!
```

Out[16]:
```
'!skcor nohtyP'
```

In [17]:
```
a[1:6:-1] # In a[i:j:-1], changes meaning of i and j
```

Out[17]:
```
''
```

Objects and Methods - A Crude Introduction

An object can be thought of a construct in the memory.

It has a well defined behavior with respect to other objects. (2*3 is allowed, "a"*"b" is not!)

The properties of the object, the operations that can be performed all are pre-defined.

A method is a function bound to an object that can perform specific operations that the object supports.

```
ObjectName.MethodName(arguments)
```

OK, let's see some string methods in action!

String Methods

In [19]:
```
a = "   I am a string, I am an object, I am immutable!   "
```

In [21]:
```
a.title()
```

Out[21]:
```
'   I Am A String, I Am An Object, I Am Immutable!   '
```

In [20]:
```
a.split(",")
```

Out[20]:
```
['   I am a string', ' I am an object', ' I am immutable!   ']
```

In [22]:
```
a.strip() # Remove trailing and leading whitespaces.
```

Out[22]:
```
'I am a string, I am an object, I am immutable!'
```

Strings are Immutable!

In [23]:
```
print a # Check the value!
```

```
I am a string, I am an object, I am immutable!
```

In [24]:
```
a.title() # Transform string to title case ... really?
```

Out[24]:
```
'   I Am A String, I Am An Object, I Am Immutable!   '
```

In [25]:
```
print a # Nothing changed! Strings are immutabe.
```

```
I am a string, I am an object, I am immutable!
```

In [26]:
```
b = a.title() # String methods return strings instead.
```

In [27]:
```
print b
```

```
I Am A String, I Am An Object, I Am Immutable!
```

In [28]:
```
a[3] = "x" # Immutability implies no in-place changes.
```

```
---------------------------------------------------------------
------------
TypeError                                 Traceback (most recen
t call last)
<ipython-input-28-b0d08958dc31> in <module>()
----> 1 a[3] = "x" # Immutability implies no in-place changes.

TypeError: 'str' object does not support item assignment
```

Getting Help

In [29]:
```
print dir(a) # a is a string object.
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__doc_
_', '__eq__', '__format__', '__ge__', '__getattribute__', '__ge
titem__', '__getnewargs__', '__getslice__', '__gt__', '__hash__
', '__init__', '__le__', '__len__', '__lt__', '__mod__', '__mul
__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__rep
r__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__s
tr__', '__subclasshook__', '_formatter_field_name_split', '_for
matter_parser', 'capitalize', 'center', 'count', 'decode', 'enc
ode', 'endswith', 'expandtabs', 'find', 'format', 'index', 'isa
lnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', '
isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'rep
lace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rst
rip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase',
 'title', 'translate', 'upper', 'zfill']
```

In [30]:
```
help(a.find)
```

```
Help on built-in function find:

find(...)
    S.find(sub [,start [,end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notatio
n.

    Return -1 on failure.
```