Installing Python

- Python is available on all platforms: Linux, MacOS & Windows
- Comes in two variants
- Python 2.7
- Python 3+ (Currently 3.6.X)
- We will be working with Python 3+

Difference between Python 2.7 & Python 3

- Python 2.7 is a "static" older version
- Many libraries for scientific & statistical computing are still available in Python 2.7, hence still in use
- Python 3 is almost identical to Python 2.7

Interpreters vs Compilers

- Programming languages like Python, C, C++, Java etc are written for us to understand and write instructions.
- Meaning programming languages are "High Level"
- Computers understand "Low Level" instructions
- Compilers, translates high level programming language to machine level instructions and generates executable code
- Interpreters, itself a program that runs and directly "understand" high level programming

Python Interpreter

- Python is an interpreted language
- First we need to invoke the interpreter
- When interpreter is running we pass python instructions/commands to the interpreter to be executed
- Interpreter is very interactive as we can play with it like a calculator.
- Can load complex programs from files

  >>> from filename import *

How a Python Program looks like?

```
def function1(….,…..,….):
def function2(….,…..,….):
.
.
def function n(….,…..,….):
…..
statement_1
statement_2
statement_3
.
.
statement_n
```

Points to Understand

- Interpreter executes statements from top to bottom
- Function definitions are digested for future use
- Actual computation starts from statement_1

Statement

- Most common statement is the assignment statement
- The equal sign (=) is used to assign a value
- Assign a value to a name

  i = 10

  j = 5*i

  j = j + 50
- In general, left hand side is a name and right hand side is an expression
- Operations on expression depends on type of value

Numeric Values

- Numbers are available in two varients

  int – integers

  float - fractional numbers
- 100, -7, 123456 are few examples of values of type integers
- 10.59, -0.09, 30.1234 are few examples of values of type float

int vs float

- Why do we have these two different types?
- Internally, a value is stored as a finite sequence of 0's and 1's (binary digits or bits)
- For an ***int,*** this sequence is read off as a binary number
- For a ***float,*** this sequence breaks up into a *mantissa* and *exponent*
- *Example:* Scientific Notation: 0.123×10^8

  Move decimal point 8 digits to the right
- Integer can be considered as a fixed decimal point but in case of float your decimal point can be considered as a floating meaning it keeps on varying depending upon exponent

NOTE: Python also has built-in support for complex numbers, and uses the j or J suffix to indicate the imaginary part (e.g. 3+5j)

Operations on Numbers

- Basic Arithmetic Operations like +, - , * , /
- Division will always produce a value of type float

  Example: 5/2 is 2.5, 8/2 is 4.0
- In general, python allows us to mix int and float

  Example: 3+1.5 = 4.5

- Floor Division and Modulus: // and %

  Example: 8//5 is 1, 8%5 is 3
- Exponentiation: **

  *Example: 9**3 is 729*

In interactive mode, the last printed expression is assigned to the variable _

Example:

```
>>> gst = 18/100
>>> gst
0.18
>>> mrp = 100
>>> mrp * gst
18.0
>>> mrp + _
118.0
```

Few Other Operations on Numbers

- log ( ), sqrt ( ), sin( )…….
- Built in python functions but are not loaded by default
- We need to include these functions explicitly by including maths library

  *from math import **

Names, Values & Types

- Names are used to remember values
- Values are those which are assigned to names.
- Main difference between python and other languages is that names themselves don't have any inherit types. As we don't mention that name is of type integer or of type float.
- Where as in other languages like C, C++, Java etc each name is declared in advance with its type.
- Names can be assigned values of different types as the program evolves

Example:

```
i = 10      # i is an integer
i = 8*2     # i is an integer
j = i/3     # j is a float
i = 2*j     # i is a float
```

- type(e) return type of expression e
- Not a Good Style!

Boolean Values

- Another important class of values which we use implicitly in our functions are boolean values which designate TRUTH or FALSE
- Typically there are three functions not, and, or which operate on these values

Example

not True is False, not False is True

m and n is True if both m,n are True

m or n is True if atleast one of m,y is True

Comparison

- Most easy way to generate Boolean values is through comparisons

  Example

  $x == y, \quad m != n$

  $a < 5*5, \quad c > d$

  $i <= j+k, \quad 20 >= 42*b$

- Boolean values can also be computed, assigned and passed around just like numerical values.

Example

```
def divides(m,n)
  if n%m == 0:
     return(True)
  else:
     return(False)

def even(n):
  return(divides(2,n))

def odd(n):
  return(not divides(2,n))
```

Manipulating Text

- Computation is lot more than number crunching
- Text processing is equally important now days

Strings – type str

- Python uses type string for text
- String is a sequence or list of characters
- Encloses in quotes – single, double and triple

Example

city = 'Chandigarh'
x = '5'
title = "Saurabh Kaushal's Python Programming Class"
myquote = ''''"Saurabh Kaushal's"''''

String as Sequences

- Characters in strings have positions
- Characters starts with position 0,1,2,3……, ***n-1***for a string of length ***n***

  Example

  x = "hello"

h	*e*	*l*	*l*	*o*

- Position -1, -2, …. Count backwards from end

  s[1] is "***e***" , s[-2] is "***l***"

Operations on Strings

- Combine two strings is called as concatenation and operator which is used for this is **+**

  s = "hello"

  t = s + ", programmers"

  t is now "hello, programmers"

- ***len(s)*** returns the length of string **"s"**

Extracting Substrings

- A slice is a "segment" of string
- s[i : j] starts at s[i] and ends at s[j – 1]
- **s** = "hello"
- s[1 : 4] is "**ell"**
- Slice is somewhat similar to range function which we saw in our gcd example
- s[: j] starts at s[0], meaning s[0 : j]

Modifying Strings

- Cannot update a string "in place"
- s = "hello", want to change to "help!"
- s[3] = "p" --- Prompt an error that 'str' object does not support item assignment
- Instead use slices and concatenation
- s = s [0 : 3] + "p!"

- Strings are ***immutable*** values meaning we cannot change them without creating a fresh value.
- ***Lists*** can be changed.