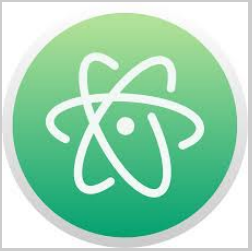
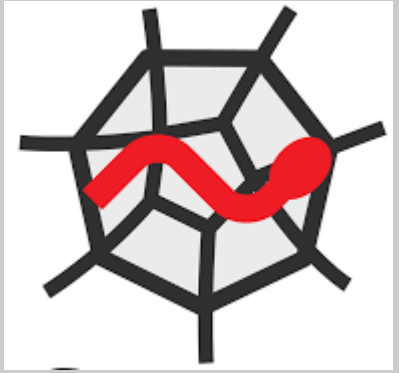
INTRO TO PYTHON WORLD

Python Programming Platforms & Pip

- **Python programs can be written on:**
- **1) Text Editors**
- **2) Python Shell**
- **3) IDE**
- **There are several packages and libraries that can be used in Python**
- **Python has 141,251+ such Packages. PyPI indexes all of those packages.**
- **These packages can be installed using PIP on a system.**
- **Pip is the package management system of python to install packages in following ways:**
- pip install <pacakage_name> // for python 2.7
- pip3 install <package_name> // for python 3 or more

Various IDE’s and Editors





Scopes of Python

Data Types in Python

- Number
- String
- List
- Tuples
- Set
- Dictionary
- Array

[One Dimensional]

- Array

[Multi Dimensional]

Number (int,float,complex,fractions)

```
>>>  complex1 = 2+3j
>>> complex2 = 4+5j
>>> complex1.imag
3
>>> complex1.real
2
>>> complex1 + complex2 ← performs complex addition
6+8j
>>> 1+2j.imag ← wrong statement if we write it like this Python thinks that we mean (1+2)j
3.0
>>> (1+2j).imag ← this is a proper way
2.0
>>> import fractions
>>> fraction1 = fractions.Fraction(1,2) ← 1/2
>>> fraction2 = fractions.Fraction(2,9) ← 2/9
>>> fraction1.numerator
1
>>> fraction1.denominator
2
>>> fraction1 + fraction2
Fraction(13, 18)
```

Strings

- A sequence of characters.
- Represented using double or single quotes.
- They are iterable
- Reverse using x[::-1]
- we can access a character in a string

eg. x = “abcd”

x[0] ← it will print a (This is indexing)

Lists

- They are iterable, mutable collection of objects
- represented using [ ] literal
- It can be Multi Dimensional

Lists inside Lists

eg. x=[[1, 2, 3, 4],

[5, 6, 7, 8],

[9, 10, 11, 12]]

- primarily 4 things we must know how to do in a list

1) adding an element. (.append(), .insert(), .extend())
2) deleting an element. *(.remove(), .pop(), .clear(), .del())
3) sorting a list (.sort(), .reverse())
4) Iterable operations (len, .count(), slicing, indexing, copying)

Tuples

- iterable and immutable collection of objects
- Homogeneous & Heterogeneous both
- represented using ( ) literal
- have only basic operations concatenation and iterable operations

Sets

- a mutable collection of unique objects
- represented using { } literal with at least one element in them
- supports the operations like difference, union, intersection
- Also check if a set is set's subset/ superset
- Also operations of list like add, pop,extend

Dictionaries

- used to establish a relation between two objects
- Mutable, Mapping object type which maps hashable objects to their values.
- created using { } literal or dict()

{ 'key1': 'value1', 'key2': 'value2' }

difference-

```
>>> set1 = {1,2,3,4}
>>> set2 = {1,3,4,5}
>>> set1 - set2
{2}
>>> set2 - set1
{5}
```

Union-

```
>>>set1.union(set2)
{1,2,3,4,5}
```

Intersection-

```
>>>set1.intersection(set2)
{1,3,4}
```

Issubset-

```
>>> set1 = {1,2,3,4}
>>> set2 = {1,2,3}
>>> set2.issubset(set1)
True ← Mind that it checks for subset not for PROPER subset.
```

Add elements-

```
>>> set1={1}
>>>set1.add(2)
>>>set1
{1, 2}
```

Remove elements-

```
>>> set1={1,2}
>>> set1.remove(2)
>>> set1
{1}
```

Pop elements-

```
>>> set1 = {1,2}
>>> set1.pop()
1
```

Accessing values in dictionary

```
>>> dictionary = {1:  2, 3: 4 , 5: 6}
>>> dictionary[1]
2
>>> dictionary[3]
4
>>> dictionary[5]
6

## Getting a list of all of the keys from a dictionary
>>> dictionary.keys()
dict_keys([1, 3, 5])

## Getting a list of all of the values from a dictionary
>> dictionary.values()
dict_values([2, 4, 6])

## Getting a list of a tuples containing (key, value)
>> dictionary.items()
dict_items([(1, 2), (3, 4), (5, 6)])
```

Mutable operations on Dictionary

Add an item

```
>>> d = {}
>>> d['x'] = 'y'
>>> d
{'x': 'y'}
```

Deleting an item

```
>>> d= {1:2, 3:4, 5:6}
>>> del(d[1])
```

Popping a value

```
>>> d= {1:2, 3:4, 5:6}
>>> d.pop(1)
2
```

Popping an item

```
>>> d= {1:2, 3:4, 5:6}
>>> d.popitem()
(5, 6)
```

Taking INPUT

- Input()
- raw_input()
- This method is not used in python 3.x versions

```
>>> x = input("Enter your name:")
Enter your name:Vaibhav Kumar
>>> print("Hello, " + x)
Hello, Vaibhav Kumar
>>>
```

```
>>> a = raw_input()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'raw_input' is not defined
```

```
>>> number = int(input("enter a number"))
enter a number23
>>> type(number)
<class 'int'>
>>>
```

Conditional Statements in Python

If statement

- IF BOOLEAN_EXPRESSION:

    STATEMENTS TO EXECUTE IF BOOLEAN_EXPRESSION IS TRUE

NOTE:-Boolean expressions are those expression which result in a boolean value

- Just like in C and Java we represent a code block using parenthesis.
- In Python we represent a block of code using indentation.
- Indentation means “some space which can be given using tabs or space character”.
- All the statements under one block of code must have same indentation.

```
>>> if True:
...     print("HELLO")
...
HELLO
>>> if False:
...   print("YES")
...
>>> if 1==1:
...    print("correct")
...
correct
>>> x = 3
>>> if x==3:
...     print("right")
...
right
>>> x=4
>>> if x==3:
...     print("right")
...
>>>
```

Conditional Statement in Python

If else statement

```
if BOOLEAN_EXPRESSION:

    STATEMENTS TO EXECUTE IF BOOLEAN_EXPRESSION IS TRUE

else:
   STATEMENTS TO EXECUTE IF BOOLEAN_EXPRESSION IS FALSE
```

Elif statement

```
if BOOLEAN_EXPRESSION1:
    CODE_BLOCK1
elif  BOOLEAN_EXPRESSION2:
    CODE_BLOCK2
elif BOOLEAN_EXPRESSION3:
    CODE_BLOCK3
else:
    CODE_BLOCK4
```

```
>>> if 1<10:
. . . .    print(“Boolean_Expression1is True So I will be printed”)
. . . .if 2<10 :
. . . .    print(“Boolean_Expression2 is also True so I will also be printed irrespective of whether 1 is true or false”)
. . . .elif 3<10:
. . . .    print(“Boolean_Expression3 is also True but will not be printed because its corresponding if is already true”)
. . . .else:
. . . .   print(“No one gives a damn about me”)
Boolean_Expression1is True So only I will be printed
Boolean_Expression2 is also True so I will also be printed irrespective of whether 1 is true or false
```

```
>>> if False:
. . . .    print("Boolean Expression was False so I will not be printed")
. . . .else:
. . . .    print("But I am inside else so as Boolean Expression is False I will get  printed")
But I am inside else so as Boolean Expression is False I will get  printed
>>> if 1==1:
. . . .    print("Because 1==1 is True so I will be printed")
. . . .else:
        print("Because 1==1 is True so I will not get printed")
Because 1==1 is True so I will be printed
```

Loops in Python

while statement

```
while BOOLEAN_EXPRESSION:
    STATEMENTS_TO_LOOP_THROUGH
```

Doing something till a condition is met or Doing something Forever but it can also Do something N times with a little tweak.

```
>>> alive=True
>>> while alive:
. . . .    print("Breathing Till Death by KeyboardInterrupt ctrl+C or ^C")
>>> stomach_full='no'
>>> while stomach_full!='yes':
. . . .    print('Eating Food')
. . . .    stomach_full = input("Is stomach full? Type yes or no in lower case only.")
Eating Food
Is stomach full? Type yes or no in lower case only.
no
Eating Food
Is stomach full? Type yes or no in lower case only.
yes
>>> cramming_done_times=1
>>> while cramming_done_times<=3:
. . . .    print(cramming_done_times, 'times Crammed Formulas')
. . . .    cramming_done_times +=1
1 times Crammed Formulas
2 times Crammed Formulas
3 times Crammed Formulas
```

Loops in Python

for statement

for iterating_variable in iterable:

    STATEMENT_TO_LOOP_THROUGH_by_iterating_variable

Do something N times or Do something to a collection of things.

Note: range(start, end, step) by default step is 1 and start is 0.

If it is given as range(some_number) then python thinks that we need the range from 0 to some_number.

like range(3) means a list as [0,1,2]

Similarly range(1,10) means [1,2,3,4,5,6,7,8,9]

```
>>> friends=['friend1', 'friend2', 'friend3']
>>> for friend in friends:
....    print('Shaking hands with', friend)
Shaking hands with friend1
Shaking hands with friend2
Shaking hands with friend3

>>> for crammed_times in range(3):
....    print(crammed_times+1, 'times formulas crammed')
1 times formulas crammed
2 times formulas crammed
3 times formulas crammed
```

```
>>> x = ['a', 'b', 'c', 'd']
>>> for index, element in enumerate(x):
....     print(element, 'is at index', index)
a is at index 0
b is at index 1
c is at index 2
d is at index 3
```

```
>>> ln= 42
>>> for guess in range(5):
...     guess = int(input())
...     if(guess == ln):
...             print("win")
...             break
... else:
...     print("loose")
...
34
23
21
25
4
loose
>>>
```

```
>>> friends=['friend1', 'friend2', 'friend3', 'friend4']
>>> for friend in friends:
....    if friend=='friend2':
....        continue
....    print('Shaking hands with', friend)
Shaking hands with friend1
Shaking hands with friend3
Shaking hands with friend4

>>> for friend in friends:
....  if friend=='friend2':
....     break
....  print('Shaking hands with', friend)
Shaking hands with friend1
```

Functions in Python

In Python we declare that some identifier is a function by suffixing a keyword ‘def” before it.

def FUNCTION_NAME(FUNCTION_ARGUMENTS):
    STATEMENTS_TO_EXECUTE (ie. a code block)

```
>>> def function_name(function_argument):
. . . .     print('I got executed')
. . . .     print('whoever called it passed this value to the function = ', function_argument)
>>> function_name('abcd')
I got executed
whoever called it passed this value to the function = abcd
>>> function_name(123)
I got executed
whoever called it passed this value to the function = 123
```

Arguments in Function

1) optional argument

def function_name(mandatory_args,optional_args=default_values):

```
>>>  def greet_user(user, do_greet=True):
. . . .      if do_greet:
. . . .           print('Hello', user)
. . . .      else:
. . . .           print("Bye", user)
>>> greet_user("Guido") ← mind that we didnt need passing value of do_greet
Hello Guido
>>> greet_user("Guido", False)
Bye Guido
```

Arguments in Function

2) positional arguments:

we are mapping values to the position of arguments in the function signature.

function(arg1=value1, arg2=value2, arg3=value3)

```
>>>  def greet_user(arg1, arg2, arg3):
. . . .      print(arg1, arg2, arg3)
>>> greet_user(1,2,3)
1 2 3
>>> greet_user(arg2=2, arg3=3, arg1=1)
1 2 3

>>> greet_user(1, arg1=1, arg2=2)

TypeError: greet_user got multiple values for argument 'arg1'
```

Lambda Functions

- One Line Functions
- single expression functions which are almost completely analogous to mathematical functions.

eg. f(x,y) = x+y can be written in lambda form as-

lambda x,y : x+y ← mind that as it is in one line there is no code block needed.

```
>>> lambda_function = lambda x : x**2
>>> lambda_function(2)
4
>>> lambda_function(3)
9
```

Some Methods of Python

- Range

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

- Syntax : range(start, stop, step)

- Split

The split() method returns a list of strings after breaking the given string by the specified separator.

- Syntax : str.split(separator, maxsplit)

Some Methods of Python

- Replace

The replace() is an inbuilt function in Python programming language that returns a copy of the string where all occurrences of a substring is replaced with another substring.

- Syntax : range(start, stop, step)

- Filter

The filter() method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.

- Syntax : filter(function, sequence)

Some Methods of Python

- Filter

It is normally used with Lambda functions to separate list, tuple, or sets

```
>>> seq = [0,1,2,3,5,8,13]
>>> result = filter(lambda x:x%2 != 0, seq)
>>> print(list(result))
[1, 3, 5, 13]
>>>
```

function: function that tests if each element of a sequence true or not.

sequence: sequence which needs to be filtered, it can be sets, lists, tuples, or containers of any iterators

Some Methods of Python

- Map( )

map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)

- Syntax: map(fun, iter)

Note: to know all built-in functions type

- dir(__builtins__)

```
>>> def myfunc(a, b):
...   return a + b
...
>>> x = map(myfunc, ('apple', 'banana', 'cherry'), ('orange', 'lemon', 'pineapple'))
>>> print(x)
<map object at 0x7fa6f245a040>
>>> #convert the map into a list, for readability:
>>> print(list(x))
['appleorange', 'bananalemon', 'cherrypineapple']
>>> #for replace method
>>> txt = "I like bananas"
>>> x = txt.replace("bananas", "apples")
>>> print(x)
I like apples
>>>
```

Try and Except Method

```
try:
    STATEMENTS_TO_TRY_TO_EXECUTE_WHICH
_MAY_PRODUCE_ERRORS
except EXCEPTION as VARIABLE_NAME_TO_HOLD_EXCEPTION:
    DO_SOMETHING_WITH_EXCEPTION
else:
    STATEMENTS_TO_EXECUTE_WHEN_EVERYTHING _WENT_FINE
finally:
    STATEMENTS_TO_ALWAYS_EXECUTE
```

```
>>> #The try block will generate an error, because x is not defined:
>>> try:
...   print(x)
... except:
...   print("An exception occurred")
...
An exception occurred
>>>
```

```
>>> try:
. . . .     1/0
. . . .except: ← Catch all exceptions
. . . .     print('Exception occured')
. . . .else:
. . . .     print('Everything went fine')
. . . .finally:
. . . .     print('I will get executed no matter what')
. . . .
Exception occured

>>> try:
. . . .     1/0
. . . . except ZeroDivisionError as e: ← Catch only ZeroDivisionError
. . . .     print(e)
. . . .
division by zero
```

Regular Expressions

- A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern.
- The Python module re provides full support for Perl-like regular expressions in Python.
-
-