Python

Scope

- Introduction to Python
- Working with Python shell
- Language Features
- Looping & Decision making structures
- Data structures
- Functions
- Modules and packages
- I/O and File handling
- Classes & Objects
- Exception handling

Agenda

- Introduction to Python
- Features
- History of Python
- Strengths & Weakness
- Installing Python
- Getting started with Python
- Working with Python shell
- Language syntax (Operators, Statements, Expressions)
- Data Types

What is Python?

- Python is a general purpose, interpreted, interactive and object oriented scripting language.

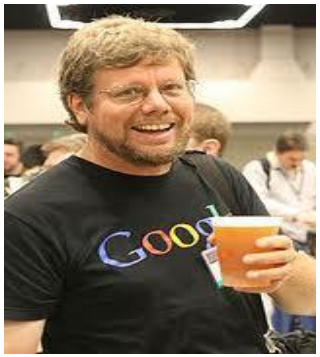
Technical Strengths of Python

- It’s Object-Oriented
- It’s free
- It’s portable
- It’s Powerful

Is Python a “Scripting Language”?

- A general-purpose programming language often applied in scripting roles.
- Commonly defined as an *object-oriented scripting language*

A bit of History



- Created by Guido Van Rossum
- In early 1990's
- @ National Research Institute for Mathematics and Computer Science in Netherlands.
- Named after the BBC show "Monty Python's Flying Circus"
- Derived from languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages
- Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Features of Python

- **Easy-to-learn, read, and maintain**
  - Few keywords, simple structure, and a clearly defined syntax
- **Easy-to-use**
  - Interactive programming experience
- **A broad standard library**
  - Portable library and cross-platform compatible on UNIX, Windows and Mac
- **Open source**
  - Free to use and distribute
- **Portable**
  - Supports a wide variety of hardware platforms

Features of Python (contd.)

- **Scalable**
  - A better structure and support for large programs
- **Extendable**
  - Support for adding low-level modules
- **Object-Oriented**
  - An object-oriented language, from the ground up with support for advanced notions as well
- **Databases**
  - Provision to connect with all major databases

What is Python useful for?

- Python is best suited for, but not limited to:
  - Systems Programming
  - Web application development – Flask, Django
  - GUI and database Programming
  - Numeric and Scientific programming
  - Throwaway programs

Who is using Python?

Installing Python

- Python distribution is available for a wide variety of platforms.
- Python is pre-installed in Linux and Mac OS machines, while installers for all the operating systems are available at https://www.python.org/downloads/

Running Python – Method 1

- **Using Interactive Interpreter:**

- Enter **python** and start coding right away in the interactive interpreter by starting it from the command line.

- You can do this from Unix, DOS or any other system, which provides you a command-line interpreter or shell window.

```
$  python                    # Unix/Linux
C:>  python                  # Windows/DOS
```

- Here is the list of all the available command line options:

Option	Description
-d	provide debug output
-O	generate optimized bytecode (resulting in .pyo files)
-S	do not run import site to look for Python paths on startup
-v	verbose output (detailed trace on import statements)
-X	disable class-based built-in exceptions (just use strings); obsolete starting with version 1.6
-c cmd	run Python script sent in as cmd string
file	run Python script from given file

Running Python – Method 2

- **Script from the Command-line:**

- A Python script can be executed at command line by invoking the interpreter on your application, as in the following:

```
$  python script.py          # Unix/Linux
C:>  python script.py        # Windows/DOS
```

- **Note:** Be sure the file permission mode allows execution.

Running Python – Method 3

- **Integrated Development Environment**

- You can run Python from a graphical user interface (GUI) environment as well.

  - **Unix:** IDLE is the very first Unix IDE for Python.
  - **Windows:** PythonWin is the first Windows interface for Python and is an IDE with a GUI. IDLE comes along with Python itself.
  - **Macintosh:** The Macintosh version of Python along with the IDLE IDE is available from the main website, downloadable as either MacBinary or BinHex'd files.

- PyDev, a plugin for Eclipse that turns Eclipse into a full-fledged Python IDE . Both Eclipse and PyDev are cross-platform and open source.

Language Basics

Identifiers

- A Python identifier is a name used to identify a variable, function, class, module, or any other object.

- **Naming Rules**

✓ Variable length can be of anything.

✓ Identifier names should start with an alphabet or underscore(_) followed by zero or more letters, underscores and digits

✓ No other special characters are allowed.

✓ Identifier names are case sensitive.

Reserved Words

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

Comments

- Comments in Python start with the hash character, #, and extend to the end of the physical line.
- A comment may appear at the start of a line or following whitespace or code, but not within a string literal.

```
# this is the first comment
SPAM = 1              # and this is the second comment
                      # ... and now a third!
STRING = "# This is not a comment."
```

- ’’’ (triple quotes) serves as multi-line comment. It can be used to generate documentation automatically.

The print Statement

- The print statement prints its argument to the output stream. Elements separated by commas print with a space between them.

```
>>> print('hello‘)
hello
>>> print('hello', 'there‘)
hello there
```

- A formatted printing can be done as below

```
'{} {}'.format('one', 'two')
```

Operators

Arithmetic Operators	Comparison Operators	Logical Operators	Assignment Operators	Bitwise Operators	Membership Operators and Identity Operators
+	>	and	=	&	in not in is is not
-	<	or	+=	|	
*	>=	not	-=	^	
/	<=		/=	~	
%	==		*=	>>	
**	!=			<<	
//					

Precedence of Operators

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Ccomplement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
== !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Python as a calculator

```
>>> 2+2
4

>>> (50-5*6)/4
5.0

>>> 8/5             # Fractions aren't lost when dividing integers
1.6

>>> 7//3            # Integer division returns the floor value
2

>>> 7//-3
-3

>>>7%3
1
```

Python as a calculator

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625

>>> x = y = z = 0              # Zero x, y and z
>>> x
0

>>> a=b=c=1
>>> a,b,c=1,2,'John'


>>>n                          # Accessing an undefined variable is ERROR
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Lines and Indentation:

- Blocks of code are denoted by line indentation. No braces or any keywords to indicate blocks of code for class and function definitions or flow control.

- The number of spaces in the indentation is variable, but all statements within the block must be indented by the same amount.

```
>>>if True:
        print("True“)
    else:
        print("False“)
```

- While the below is error

```
>>>if True:
        print("Answer“)
        print("True" )
     else:
                print("Answer“)
            print("False“)
```

Multi Line statements

- Example of a multi line statement is

```
>>> total = item_one + \
      item_two + \
      item_three
```

- Statements contained within the [], {} or () brackets do not need to use the line continuation character.

- For example:

```
days = ['Monday', 'Tuesday', 'Wednesday',
                  'Thursday', 'Friday']
```

- On the other hand, to use multiple statements in a single line, ; is to be used.

Data Types

- Python supports the below standard data types:
  - Numbers
  - String
  - Boolean
  - List
  - Tuple
  - Set
  - Dictionary

Python Numbers

- Number data types store numeric values. They are immutable data types which means that changing the value of a number data type results in a newly allocated object.

- Number objects are created when one assigns a value to them.

```
>>> var1 = 1
>>> var2 = 10
```

- To delete the reference to a number object using the **del** statement.

```
>>> del var
>>> del var_a, var_b
```

Python Numbers

- int (Integers – No limit to the value of integers)
  - Octal Ex) 0o12
  - Hexadecimal Ex) 0xF
  - Binary Ex) 0b0011
- float (floating point values in scientific or exponential form)
- complex (complex numbers)

int	float	complex
10	0.0	3.14j
100	15.20	45.j
-786	-21.9	9.322e-36j
080	32.3+e18	.876j
-0490	-90.	-.6545+0J
-0x260	-32.54e100	3e+26J
0x69	70.2-E12	4.53e-7j

Complex numbers

```
>>> (0+1j) * (0+1J)
(-1+0j)

>>> 1j * complex(0, 1)
(-1+0j)

>>> 3+1j*3
(3+3j)

>>> (3+1j)*3
(9+3j)

>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

Strings

- Strings can be enclosed in single, double or triple quotes.
- Usually the single quote for a word, double quote for a line and triple quote for a paragraph.

```
>>>word = 'word'
>>>sentence = "This is a sentence."
>>>paragraph = """This is a \
paragraph \
across \
multiple lines."""
```

- Starting with Python 3.0 all strings support Unicode

Strings

- More examples on strings

>>> ’spam eggs’
’spam eggs’

>>> ’doesn**\’**t’
"doesn’t“

>>> "doesn’t"
"doesn’t“

>>> ’"Yes," he said.’
’"Yes," he said.’

>>> "**\"**Yes,**\"** he said."
’"Yes," he said.’

>>> ’"Isn**\’**t," she said.’
’"Isn\’t," she said.’

Raw String

- **Raw string** literals, with an "r" prefix, escape any escape sequences within them

```
>>> print('C:\some\name')
C:\some
ame

>>> print(r'C:\some\name')
C:\some\name
```

Python Strings

```
>>>str = 'Hello World!'

>>>str                  # Prints complete string
>>>str[0]               # Prints first character of the string
>>>str[2:5]             # Prints characters starting from 3rd to 5th
>>>str[2:]              # Prints string starting from 3rd character
>>>str[-1]              # Prints the last item from the end
>>>str * 2              # Prints string two times
>>>str + "TEST"         # Prints concatenated string
```

String Operations

```
>>>s = “Python“
>>>len(s)                  # Length of the string
6

>>>s.find(‘t’)             # Use “find” to find the start of a substring.
2

>>>s.replace(‘P', ‘J')     # Replace a substring with another
Jython

>>>s.upper()               # Change to upper case
PYTHON

>>>s='aaa,bbb,ccc,dd’
>>>s.split(“,")            # Split the string into parts using  ‘,’ as delimiter
['aaa','bbb','ccc','dd']

>>>s.isalpha()             # Content tests: isalpha, isdigit, etc.
True
```

String Operations

```
>>>s = ‘aaa,bbb,ccc,dd  \n‘

>>>s.rstrip()                    # Remove whitespace characters on the right
aaa,bbb,ccccc,dd

>>> line.startswith(“a")        # Check if the string starts with ‘a’
True

>>> line.endswith(“c")          # Check if the string ends with ‘c’
False

>>> names = [“Ben", “Hen", “Pen"]

>>> ", ".join(names)            # Join the list elements into a string using ‘,’
‘Ben, Hen, Pen’

>>> "Br" in “Brother”           # ‘in’ and ‘not in’ operators to check the existence
True
```

Strings are immutable

- Strings are read only

```
>>> s = “python"
>>> s[0] = “P"
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment

>>> s = “P" + s[1:]
>>> s
‘Python‘
```

Python Lists

- Ordered collection of arbitrary elements with no fixed size.

```
>>>list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
>>>tinylist = [123, 'joy']

>>>list                # Prints complete list
>>>list[0]             # Prints first element of the list
>>>list[1:3]           # Prints elements starting from 2nd till 3rd
>>>list[2:]            # Prints elements starting from 3rd element
>>>list[:3]            # Prints elements starting from beginning till 3rd
>>>list[1:-1]          # Prints all elements except the first and last
>>>tinylist * 2        # Prints list two times
>>>list + tinylist     # Prints concatenated lists
>>>len(list)           # Prints length of the list
```

Modifying Lists

- List is mutable.

```
>>> list[0]=111          # Changes the first element to 111
>>> a[2:4]=[20,30]       # Changes the third and fourth elements
>>> a[2:4]=[]            # Removes the third and fourth elements
>>> a[:]=[]              # Clears the list
```

Nesting of Lists

- It is possible to nest lists (create lists containing other lists), for example:

```
>>> q = [2, 3]
>>> p = [1, q, 4]
>>> len(p)
3
>>> p[1]
[2, 3]
>>> p[1][0]
2
```

Adding to Lists

- Method 1 : Using list concatenation

```
>>>a=a+[10,20]
```

This will create a second list in memory which can (temporarily) consume a lot of memory when you're dealing with large lists

- Method 2 : Using append

append takes a single argument(any datatype) and adds to the end of list

```
>>>a=[10,20,30]
>>>a.append('new')
>>> a
[10,20,30,'new']
>>> a.append([1,2,3])
>>>a
[10,20,30,'new',[1,2,3]]
```

Adding to Lists

- Method 3 : Using extend

extend takes a single argument(list),and adds each of the items to the list

```
>>>a=[10,20,30]
>>>a.extend([1,2,3])
>>>a
[10,20,30,1,2,3]
```

- Method 4 : Using insert

Insert can be used to insert an item in the desired place

```
>>>a=[10,20,30]
>>>a.insert(0,‘new’)
>>> a
[’new’,10,20,30]
>>>a.insert(100,‘python’)
>>> a
[’new’,10,20,30,’python’]
```

Deletion in Lists

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]

>>> a.remove(333)          # removes the first matching value
>>> a
[66.25, -1, 333, 1, 1234.5, 333]


>>> a = [-1, 1, 66.25, 333, 333, 1234.5]

>>>a.pop(2)               # returns the removed element whose in index is given
66.25
>>>a
[-1, 1, 333, 333, 1234.5]
```

Deletion in Lists

```
>>> del a[0]                    # removes an item in the specificied index
>>> a
[1, 66.25, 333, 333, 1234.5]

>>> del a[2:4]
>>> a
[1, 66.25, 1234.5]

>>> del a[:]
>>> a
[]

>>>del a
>>>a
Name Error : name ‘a’ is not defined
```

More on Lists

```
>>> a = [66.25, 333, 333, 1, 1234.5]

>>> print(a.count(333), a.count(66.25), a.count('x'))
2 1 0

>>> a.index(333)              # Returns the index of the  given value in the list
1

>>>a.index(1000)              # Error if the value is not in the list
VALUE ERROR

>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]

>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

List Comprehensions

```
>>> squares = []

>>> for x in range(10):
        ... squares.append(x**2)

>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

We can obtain the same result with List comprehension as below:

```
>>>squares = [x**2 for x in range(10)]
```

zipping lists together

```
>>> names
['ben', 'chen', 'yaqin']

>>> gender = [0, 0, 1]

>>> list(zip(names, gender))
[('ben', 0), ('chen', 0), ('yaqin', 1)]
```

NOTE : Additional elements without match will be ignored

Python Tuples

- A tuple consists of a number of values separated by commas.
- Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are:
  - Lists are enclosed in brackets ([ ]) and their elements and size can be changed
  - Tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as **read-only** lists.

Python Tuples

```
>>>tuple = ( 'abcd', 786 , 2.23, 'joy', 70.2  )
>>>tinytuple = (123, 'joe')

>>>tuple                    # Prints complete list
>>>tuple[0]                 # Prints first element of the list
>>>tuple[1:3]               # Prints elements starting from 2nd till 3rd
>>>tuple[2:]                # Prints elements starting from 3rd element
>>>tinytuple * 2            # Prints list two times
>>>tuple + tinytuple        # Prints concatenated lists

>>>tuple[2] = 1000          # Invalid syntax with tuple
   Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
   TypeError: 'tuple' object does not support item assignment

>>>list[2] = 1000           # Valid syntax with list
```

Python Sets

- A set is an unordered collection with no duplicate elements.
- Basic uses include membership testing and eliminating duplicate entries.
- Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.
- Curly braces or the set() function can be used to create sets

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}

>>> print(basket)                # show that duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket           # fast membership testing
True
>>> 'crabgrass' in basket
False
```

Python Sets

```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                          # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                      # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b                      # letters in either a or b
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                      # letters in both a and b
{'a', 'c'}
>>> a ^ b                      # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

Python Dictionary

- Python's dictionaries are kind of hash table type.
- They work like associative arrays or hashes found in Perl and consist of key-value pairs.
- A dictionary key can be almost any Python type, but are usually numbers or strings.
- Values can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- Dictionaries have no concept of order among elements.

Python Dictionary

```
>>> D = {}

>>> D['name'] = 'Bob'        # Create key-value pairs by assignment
>>> D['job'] = 'dev'
>>> D['age'] = 40

>>> D
{'age': 40, 'job': 'dev', 'name': 'Bob'}
>>> D[‘name']
Bob

>>> tinydict = {'name': 'john', 'dept': 'sales'}
>>> tinydict.keys()          # Prints all the keys
>>> tinydict.values()        # Prints all the values
>>> tinydict
{'name': 'john‘}
```

del in Dictionary

```
>>> d
{1: 'hello', 2: 'there', 10: 'world'}
>>> del(d[2])
>>> d
{1: 'hello', 10: 'world'}
```

Data Type Conversion:

Function	Description
int(x [,base])	Converts x to an integer. base specifies the base if x is a string.
long(x [,base])	Converts x to a long integer. base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.
oct(x)	Converts an integer to an octal string.

Input

- The input(string) method returns a line of user input as a string
- The parameter is used as a prompt
- The string can be converted by using the conversion methods int(string), float(string), etc.

Input: Example

```
print("What's your name?“)
name = input("> ")

print("What year were you born?“)
birthyear = int(input("> "))

print("Hi %s! You are %d years old!" % (name, 2015 - birthyear))
```

```
~: python input.py
What's your name?
> Michael
What year were you born?
>1980
Hi Michael! You are 35 years old!
```

Decision Making

- Decision making structures allow programmers to test one or more conditions and take actions accordingly.

Statement	Description
if statements	An if statement consists of a boolean expression followed by one or more statements.
if...else statements	An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
nested if statements	You can use one if or else if statement inside another if or else if statement(s).

```
var = 100

if ( var  == 100 ) :
            print("Value of expression is 100“)

print("Good bye!“)
```

Decision Making

```
var1 = 100
if var1:
   print("1 - Got a true expression value“)
   print(var1)
else:
   print(“0 - Got a false expression value“)
   print(var1)

var2 = 0
if var2:
   print("2 - Got a true expression value“)
   print(var2)
else:
   print(“0 - Got a false expression value“)
   print(var2)
```

Decision Making

```
var = 100
if var == 200:
   print("1 - Got a true expression value“)
   print(var)
elif var == 150:
   print("2 - Got a true expression value“)
   print(var)
elif var == 100:
   print("3 - Got a true expression value“)
   print(var)
else:
            print("4 - Got a false expression value“)
            print(var)
```

Decision Making

```
var = 100
if var < 200:
   print("Expression value is less than 200“)
   if var == 150:
      print("Which is 150“)
   elif var == 100:
      print("Which is 100“)
   elif var == 50:
      print("Which is 50“)
   else:
      print(“none of the above”)

elif var < 50:
   print("Expression value is less than 50“)
else:
   print("Could not find true expression“)
```

Range Test

```
if (3 <= Time <= 5):
    print(“Office Hour“)
```

Loops

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

While Loop

```
count = 0
while (count < 9):
   print('The count is:', count)
   count = count + 1
```

```
count = 0
while count < 5:
   print(count, " is  less than 5“)
   count = count + 1
else:
   print(count, " is not less than 5“)
```

For Loop

```
for letter in 'Python':                    # First Example
   print('Current Letter :', letter)

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:                       # Second Example
   print('Current fruit :', fruit)


for num in range(10,20):                   # to iterate between 10 to 20
   for i in range(2,num):                  # to iterate on the factors of the number
      if num%i == 0:                       # to determine the first factor
         j=num/i                           # to calculate the second factor
         print('%d equals %d * %d' % (num,i,j))
         break                    #to move to the next number, the #first FOR
   else:                          # else part of the loop
      print(num, 'is a prime number‘)
```

Nested Loop

```
i = 2
while(i < 100):
   j = 2
   while(j <= (i/j)):
      if not(i%j): break
      j = j + 1
   if (j > i/j) : print(i, " is prime“)
   i = i + 1
```

More Looping techniques

- When looping through dictionaries, the key and corresponding value can be retrieved at the same time using the items() method.

```
>>> fruits= {’apple’: ’red’, ‘mango’: ’yellow’}
>>> for k, v in fruits.items():
... print(k, v)
...

apple red
mango yellow
```

More Looping techniques

- When looping through a sequence, the position index and corresponding value can be retrieved at the same time using the enumerate() function.

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
... print(i, v)
...

0 tic
1 tac
2 toe
```

More Looping techniques

- To loop over two or more sequences at the same time, the entries can be paired with the zip() function.

```
>>> questions = [’name’, ’quest’, ’favorite color’]
>>> answers = [’lancelot’, ’the holy grail’, ’blue’]
>>> for q, a in zip(questions, answers):
... print(’What is your {0}? It is {1}.’.format(q, a))
...
```

```
What is your name? It is lancelot.
What is your quest? It is the holy grail.
What is your favorite color? It is blue.
```

More Looping techniques

- To loop over a sequence in reverse, first specify the sequence in a forward direction and then call the reversed() function.

```
>>> for i in reversed(range(1, 10, 2)):
        print(i)

9
7
5
3
1
```

More Looping techniques

- To loop over a sequence in sorted order, use the sorted() function which returns a new sorted list while leaving the source unaltered.

```
>>> basket = [’apple’, ’orange’, ’apple’, ’pear’, ’orange’, ’banana’]
>>> for f in sorted(basket):
        print(f)

apple
banana
orange
pear
```

Loop control statement - break

```
for letter in 'Python':     # First Example
   if letter == 'h':
      break
   print('Current Letter :', letter)

var = 10                    # Second Example
while var > 0:
   print('Current variable value :', var)
   var = var -1
   if var == 5:
      break
```

Loop control statement - continue

```
for letter in 'Python':     # First Example
   if letter == 'h':
      continue
   print('Current Letter :', letter)

var = 10                    # Second Example
while var > 0:
   var = var -1
   if var == 5:
      continue
   print('Current variable value :', var)
print("Good bye!“)
```

Loop control statement - pass

The **pass** statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

The **pass** statement is a *null* operation; nothing happens when it executes.

The **pass** is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example):

```
for letter in 'Python':
   if letter == 'h':
      pass
      print('This is pass block‘)
   print('Current Letter :', letter)
```

Summary

- The following topics are covered so far
  - Introduction to Python
  - Features
  - History of Python
  - Strengths & Weakness
  - Installing Python
  - Getting started with Python
  - Working with Python shell
  - Language syntax (Operators, Statements, Expressions)
  - Data Types
  - Control Structures

Thank you