# Introduction to Python

Avinash Pandey

Department of Computer Science
Jaypee Institute of Information Technology, Noida

11 - 15 June, 2018

## Outline

## Brief History of Python

- Python is a programming language created in 1991 by Guido van Rossum.
- Open source general-purpose language as well as scripting language
- Object Oriented
- Interpreted
- Dynamically typed
- Increasingly popular

## Outline

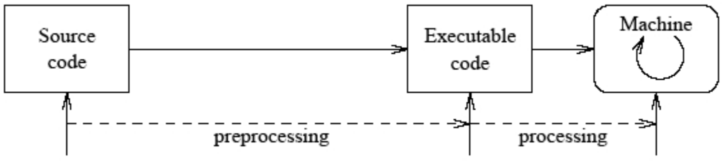
## Why Python

- Open source language
- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
  - Most of Linux and Mac versions have Python pre-installed.
- Python has a simple syntax similar to the english language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

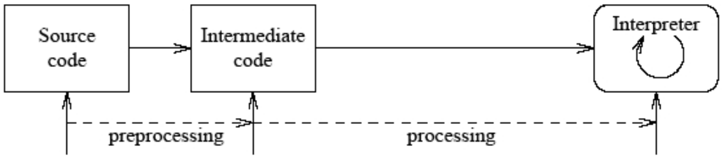
## Python v/s Other Languages

- On an average Python code is smaller than JAVA/C++ codes by 3.5 times owing to Pythons built in datatyeps and dynamic typing.
- Dynamically declare and use variables
- Python is interpreted while C/C++, etc. are compiled.
  - Compiler : spends a lot of time analyzing and processing the program, faster program execution
  - Intrepreter : relatively little time is spent analyzing and processing the program, slower program execution

## Compiler v/s Interpreter



Compilation



Interpretation

## Outline

## Who Uses Python

- Google makes extensive use of Python in its web search system
- Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, and IBM use Python for hardware testing
- YouTube video sharing service is largely written in Python
- NASA uses Python for hardware testing and satellite launching
- ...the list goes on...

## Outline

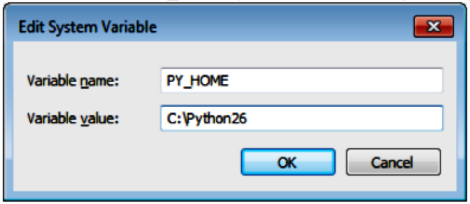
## Python Releases

- Python 1.0 released in 1994
- Python 2.0 released in 2000
- Python 3.0 released in 2008
- Python 2.7 released in 2010
- Python 3.6.0 released in 2016

Python for Windows-
https://www.python.org/downloads/
For the session we will stick to Python 3.0 and above version

## Setting pythonpath in windows

1. [Right Click]Computer > Properties >Advanced System Settings > Environment Variables
2. Click [New] under "System Variable"
3. Variable Name: PY_HOME, Variable Value:C:\path\to\python\version



Edit System Variable

Variable name: PY_HOME

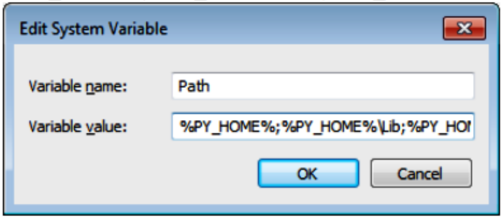
Variable value: C:\Python26

4. Click [OK]

## Setting pythonpath in windows

5. Locate the "Path" System variable and click [Edit]
6. Add the following to the existing variable:

%PY_HOME%;%PY_HOME%\Lib;%PY_HOME%\DLLs;%PY_HOME%\Lib\lib-tk;


Edit System Variable

Variable name: Path

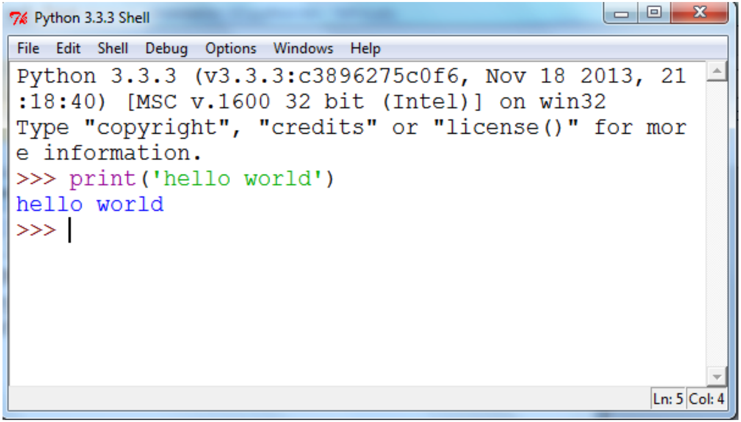
Variable value: %PY_HOME%;%PY_HOME%\Lib;%PY_HOI

7. Click [OK] to close all of the windows.

## Python first program

**Print Hello world**

1. Using Python GUI (IDLE)

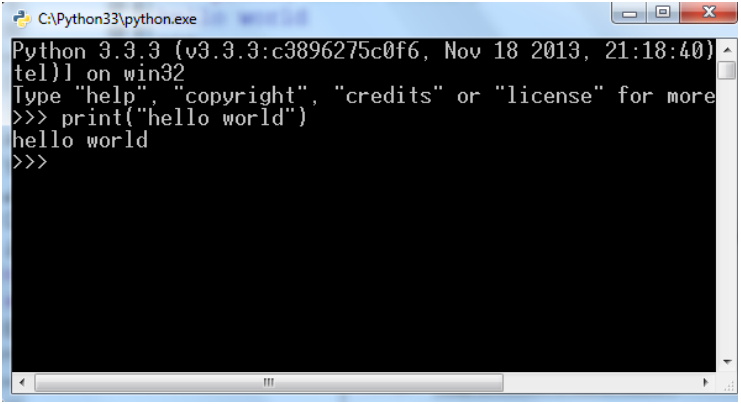


```
Python 3.3.3 (v3.3.3:c3896275c0f6, Nov 18 2013, 21
:18:40) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for mor
e information.
>>> print('hello world')
hello world
>>> 
```

## Python first program

**Print Hello world**

2. Using Python command line

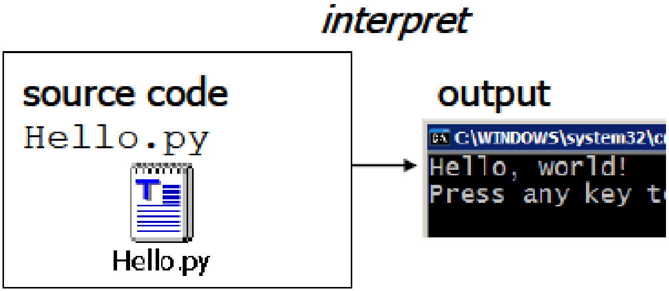


```
Python 3.3.3 (v3.3.3:c3896275c0f6, Nov 18 2013, 21:18:40)
tel)] on win32
Type "help", "copyright", "credits" or "license" for more
>>> print("hello world")
hello world
>>>
```

## Python first program

**Print Hello world**

- create new file in text editor
- Write the required statement(s).
- Save file with .py extension
- Run the program



```
Hello, world!
Press any key t
```

## Outline

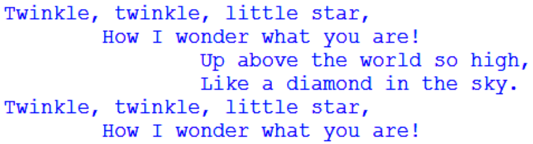
## print in python

- Print method prints the given text message or expression value on the console, and moves the cursor down to the next line.
  **syntax**
  -print('Write any message')
- Prints several messages and/or expressions on the same line.
  -print (Item1, Item2, ..., ItemN)
- Output formatting
  -print('The value of x is {} and y is {}'.format(5,10))
  Here the curly braces {} are used as placeholders. We can specify the order in which it is printed by using numbers (tupleindex).
- C-like syntax can also be used.
  -print("%s %s" %('hello', 'world'))

## Outline

## Excercise 1

- **WAP to print your name two times**
- **WAP to add three numbers and print result**
- **WAP to print the following string in a specific format**
  Sample String : *"Twinkle, twinkle, little star, How I wonder what you are! Up above the world so high, Liek a diamond in the sky. Twinkle, twinkle, little star, How I wonder what you are"*
  **Output :**



```
Twinkle, twinkle, little star,
        How I wonder what you are!
                Up above the world so high,
                Like a diamond in the sky.
Twinkle, twinkle, little star,
        How I wonder what you are!
```

## Outline

## Comments in python

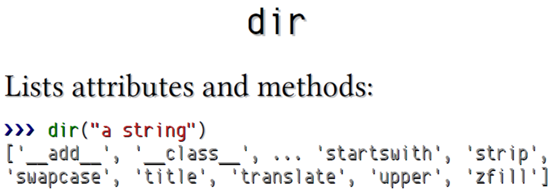
- For single line comment hash (#) symbol is used.
- For multi-line comments use triple quotes, either ''' or """.
  """This is also a
  perfect example of
  multi-line comments"""

## Outline

## Help in python

```
Python 3.3.3 (v3.3.3:c3896275c0f6, Nov 18 2013, 21:18:40) [MSC v.1600 32 bit
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> help()

Welcome to Python 3.3!  This is the interactive help utility.

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/3.3/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules",
"keywords", or "topics".  Each module also comes with a one-line summary
of what it does; to list the modules whose summaries contain a given word
such as "spam", type "modules spam".

help> string
Help on module string:

NAME
    string - A collection of string constants.
```

## dir in python



dir

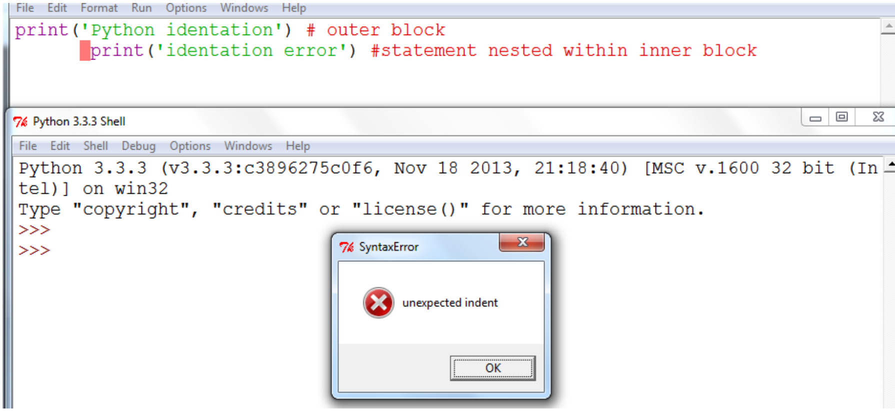
Lists attributes and methods:

```
>>> dir("a string")
['__add__', '__class__', ... 'startswith', 'strip',
'swapcase', 'title', 'translate', 'upper', 'zfill']
```

## Python indentation

- Python uses indentation instead of braces to determine the scope of expressions.
- All lines must be indented the same amount to be part of the scope (or indented more if part of an inner scope).
- This forces the programmer to use proper indentation since the indenting is part of the program.

## Python indentation


File Edit Format Run Options Windows Help
Python 3.3.3 Shell
File Edit Shell Debug Options Windows Help
SyntaxError
OK

```
print('Python identation') # outer block
       print('identation error') #statement nested within inner block
```

```
Python 3.3.3 (v3.3.3:c3896275c0f6, Nov 18 2013, 21:18:40) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
>>>
```

unexpected indent

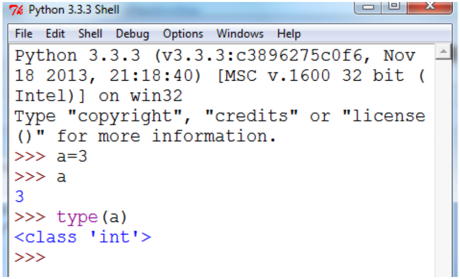
## Variables in python

- Python variables are not declared, just assigned

  **Syntax**

  - name= value

  - x=3



```
Python 3.3.3 (v3.3.3:c3896275c0f6, Nov
18 2013, 21:18:40) [MSC v.1600 32 bit (
Intel)] on win32
Type "copyright", "credits" or "license
()" for more information.
>>> a=3
>>> a
3
>>> type(a)
<class 'int'>
>>>
```

- The variable is created the first time you assign it a value.

## Variables in python

- Everything in Python is an object that has:
  - an identity (id)
    The id() function returns identity of the object (an integer value) which is unique for given object and remains constant during its lifetime.
    -id(5)
    6406896
  - a value (mutable or immutable)
- Python variables are references to objects.
- In Python, multiple assignments can be made in a single statement as follows:
  - a, b, c = 5, 3.2, "Hello"

## Variables in python

- **Mutable:** Mutable object can be changed after it is created. When you alter the item, the id will be still the same. Dictionary, List
  - $b = []$
  - $id(b) = 140675605442000$
  -$b.append(3)$
  -$b = [3]$
  - $id(b)$= 140675605442000 #$SAME$!
- **Immutable:** Immutable object can not be changed after it is created. String, Integer, Tuple
- - a = 4
  - id(a) = -6406896
  -a = a + 1
  -id(a) = -6406872 # DIFFERENT!

## Naming rules in Python

- Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.
  - *bob*, *_bob*, *simple_interest*, etc.
- Reserved words such as break, for, continue, etc. can't be used as variable's names.

## Outline

## User Input in Python

- 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.

## User Input in Python

```
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, 2011 - birthyear))
```

```
~: python input.py
What's your name?
> Michael
What year were you born?
>1980
Hi Michael! You are 31 years old!
```

## Outline

## Excercise 2

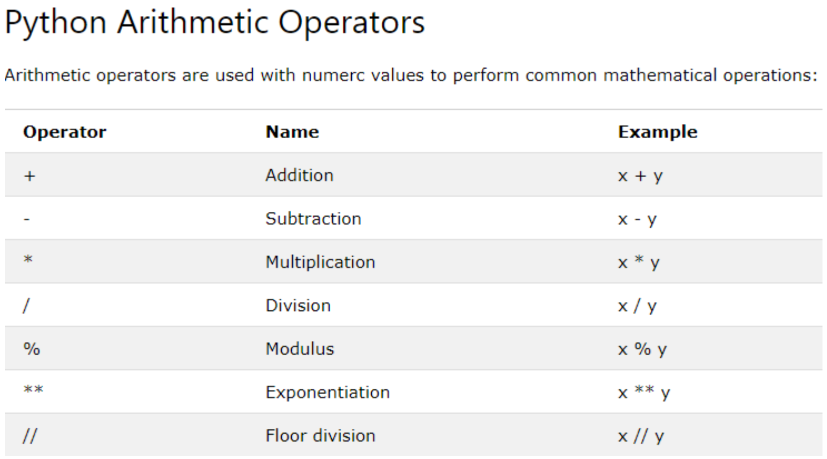
- Write a Python program which accepts the radius of a circle from the user and compute the area.
- Write a Python program which accepts the user's first and last name and print them in reverse order with a space between them.
- WAP to concatinate three strings and print result

## Outline

## Python basic operators

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

## Arithmetic operators

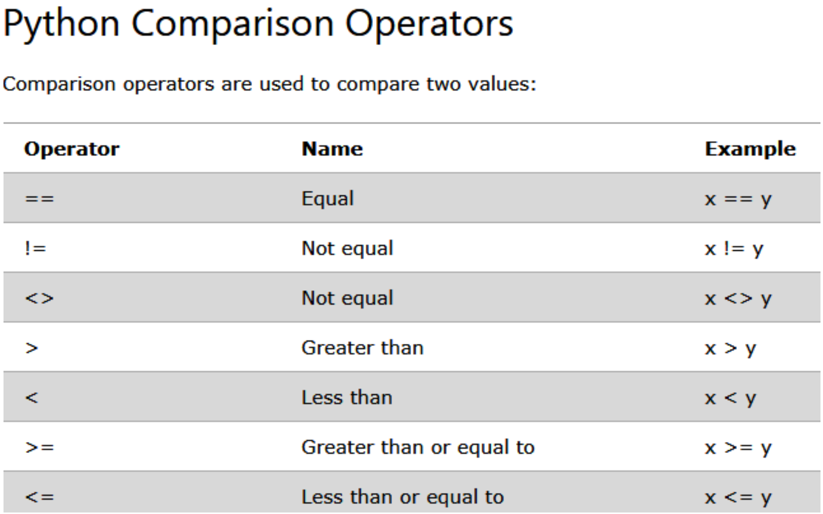


### Python Arithmetic Operators

Arithmetic operators are used with numerc values to perform common mathematical operations:

| Operator | Name | Example |
|---|---|---|
| + | Addition | x + y |
| - | Subtraction | x - y |
| * | Multiplication | x * y |
| / | Division | x / y |
| % | Modulus | x % y |
| ** | Exponentiation | x ** y |
| // | Floor division | x // y |

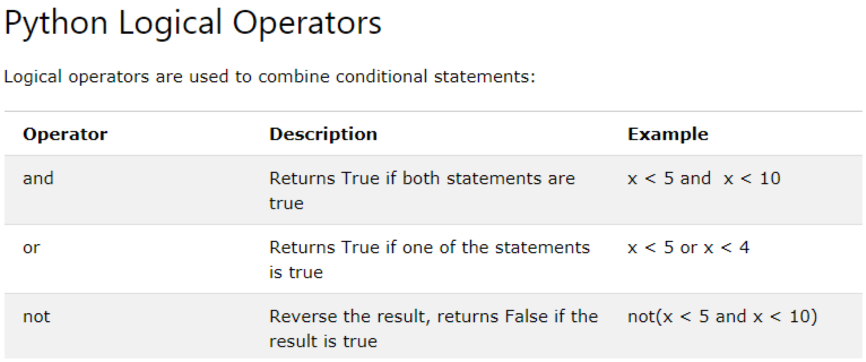
## Relational operators



### Python Comparison Operators

Comparison operators are used to compare two values:

| Operator | Name | Example |
|---|---|---|
| == | Equal | x == y |
| != | Not equal | x != y |
| <> | Not equal | x <> y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |



## Python Logical Operators

Logical operators are used to combine conditional statements:

| Operator | Description | Example |
|---|---|---|
| and | Returns True if both statements are true | x < 5 and  x < 10 |
| or | Returns True if one of the statements is true | x < 5 or x < 4 |
| not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10) |

## Membership operators

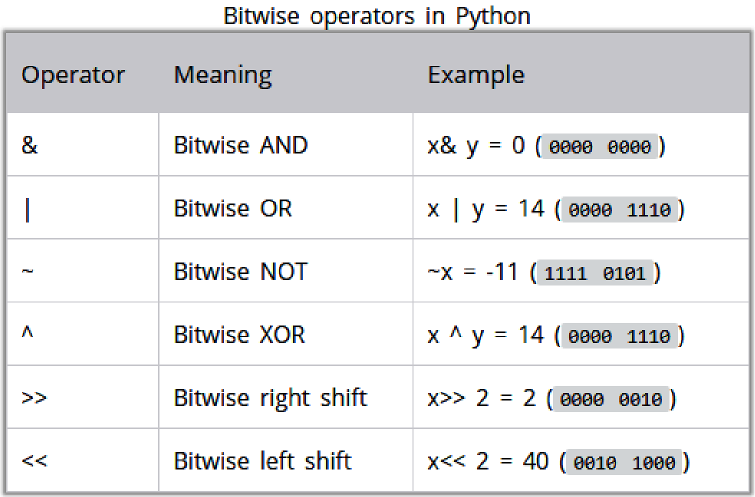
- **in** and **not in** are the membership operators in Python. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

```
x = 'Hello world'
y = 1:'a',2:'b'
# Output: True
print('H' in x)
# Output: True
print('hello' not in x)
# Output: True
print(1 in y)
# Output: False
```

## Identity operators

- **is** and **is not** are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory.
  ```
  x = ["apple", "banana"]
  y = ["apple", "banana"]
  z = x
  print(x is z)
  # returns True because z is the same object as x
  print(x is y)
  # returns False because x is not the same object as y, even if they have thew same content
  ```

## Bitwise operators



Bitwise operators in Python

| Operator | Meaning | Example |
|---|---|---|
| & | Bitwise AND | x& y = 0 ( `0000 0000` ) |
| \| | Bitwise OR | x \| y = 14 ( `0000 1110` ) |
| ~ | Bitwise NOT | ~x = -11 ( `1111 0101` ) |
| ^ | Bitwise XOR | x ^ y = 14 ( `0000 1110` ) |
| >> | Bitwise right shift | x>> 2 = 2 ( `0000 0010` ) |
| << | Bitwise left shift | x<< 2 = 40 ( `0010 1000` ) |

## Basic Datatypes

- Integers (default for numbers).
  - z=5

  Range of int in python is equivalent of a C long

  Long Integer an unbounded integer value.
- Floats
  - z=5.2
- Strings - strings are enclosed within either single quotes or (' ') or double quotes (“ ”).

  z='python'

  z="python"

  - Use triple double quotes for multi line strings or strings than contain both ' and " inside of them:

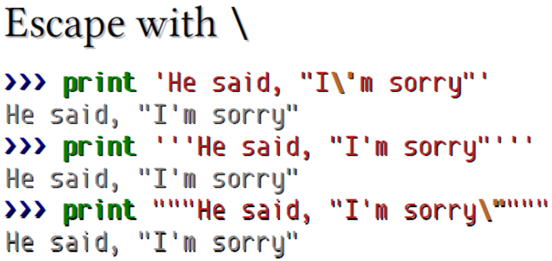
  “““python is ' open source

  " language"

  ”””

## Outline

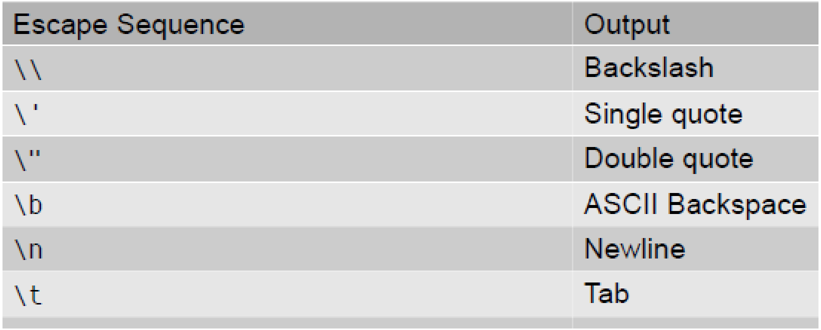
## String Escaping



Escape with \

```
>>> print 'He said, "I\'m sorry"'
He said, "I'm sorry"
>>> print '''He said, "I'm sorry"'''
He said, "I'm sorry"
>>> print """He said, "I'm sorry\""""
He said, "I'm sorry"
```

## String Escaping



| Escape Sequence | Output |
|---|---|
| \\ | Backslash |
| \' | Single quote |
| \" | Double quote |
| \b | ASCII Backspace |
| \n | Newline |
| \t | Tab |

## Strings

Characters in a string are numbered with *indexes* starting at 0:

- Example:

```
name = "P. Diddy"
```

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| character | P | . | | D | i | d | d | y |

Accessing an individual character of a string:

***variableName*** [ ***index*** ]

- Example:

```
print name, "starts with", name[0]
```

Output:

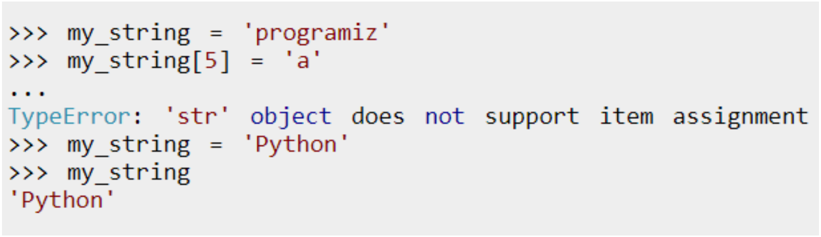
```
P. Diddy starts with P
```

```
>>> s = '012345'
>>> s[3]
'3'
>>> s[1:4]
'123'
>>> s[2:]
'2345'
>>> s[:4]
'0123'
>>> s[-2]
'4'
```

## Strings

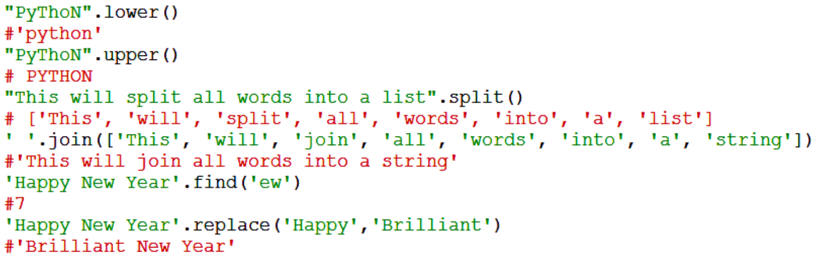
### Change or delete a String

Strings are immutable. This means that elements of a string cannot be changed once it has been assigned. We can simply reassign different strings to the same name.



```
>>> my_string = 'programiz'
>>> my_string[5] = 'a'
...
TypeError: 'str' object does not support item assignment
>>> my_string = 'Python'
>>> my_string
'Python'
```

## Python String Methods



```
"PyThoN".lower()
#'python'
"PyThoN".upper()
# PYTHON
"This will split all words into a list".split()
# ['This', 'will', 'split', 'all', 'words', 'into', 'a', 'list']
' '.join(['This', 'will', 'join', 'all', 'words', 'into', 'a', 'string'])
#'This will join all words into a string'
'Happy New Year'.find('ew')
#7
'Happy New Year'.replace('Happy','Brilliant')
#'Brilliant New Year'
```

## Basic data types
### Complex

- Complex
  - Built into Python
  - Same operations are supported as integer and float
  x = 3 + 2j
  y = -1j
  x + y will print (3+1j)

## Outline

## Python List

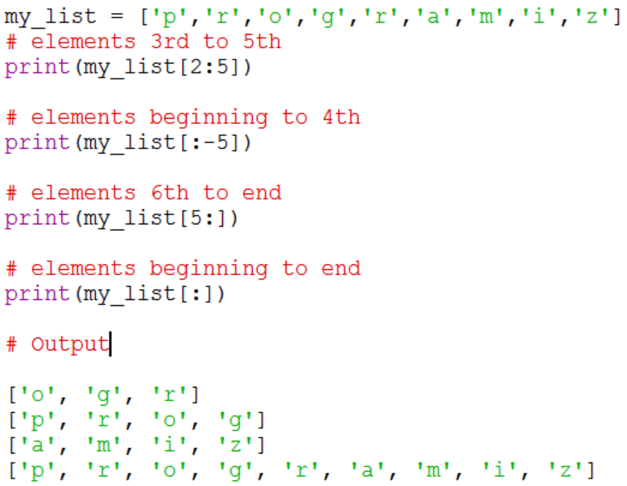
List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.

```
# empty list
mylist = []
# list of integers
mylist = [1, 2, 3]
# list with mixed datatypes
mylist = [1, "Hello", 3.4]
# nested list
mylist = ["mouse", [8, 4, 6], ['a']]
```

## Accessing list elements

# list with mixed datatypes
mylist = [1, "Hello", 3.4]
-mylist[0] will return 1
# nested list
mylist = ["mouse", [8, 4, 6], ['a']]
mylist [0][0] return m
mylist [0] return mouse

## Slicing operator for python list



```
my_list = ['p','r','o','g','r','a','m','i','z']
# elements 3rd to 5th
print(my_list[2:5])

# elements beginning to 4th
print(my_list[:-5])

# elements 6th to end
print(my_list[5:])

# elements beginning to end
print(my_list[:])

# Output

['o', 'g', 'r']
['p', 'r', 'o', 'g']
['a', 'm', 'i', 'z']
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
```

## Add elements to a list

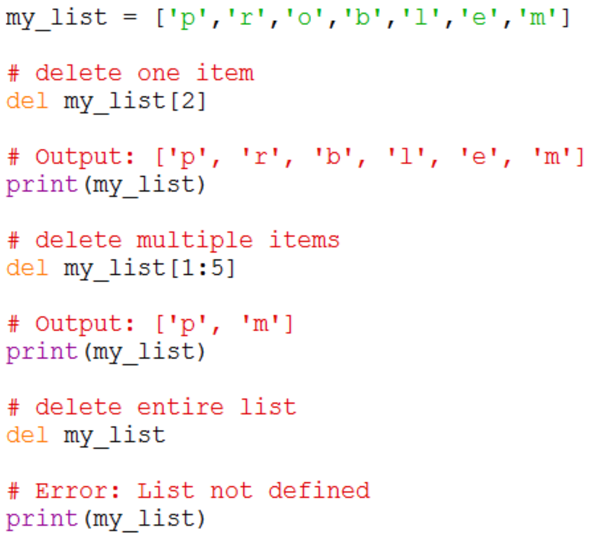
```
a1 = [1, 'x', 'y']
a2 = [1, 'x', 'y']
a3 = [1, 'x', 'y']
b = [2, 3]
a1.append(b)
a2.insert(3, b)
a3.extend(b)
print(a1)
print(a2)
print(a3)

# output
[1, 'x', 'y', [2, 3]]
[1, 'x', 'y', [2, 3]]
[1, 'x', 'y', 2, 3]
```

## Add elements to a list

To combine two lists + operator can also be used. The * operator repeats a list for the given number of times.

```
odd = [1, 3, 5]
print(odd + [9, 7, 5])
#Output: [1, 3, 5, 9, 7, 5]
print(["re"] * 3)
#Output: ["re", "re", "re"]
```

## Delete elements from a list



```python
my_list = ['p','r','o','b','l','e','m']

# delete one item
del my_list[2]

# Output: ['p', 'r', 'b', 'l', 'e', 'm']
print(my_list)

# delete multiple items
del my_list[1:5]

# Output: ['p', 'm']
print(my_list)

# delete entire list
del my_list

# Error: List not defined
print(my_list)
```

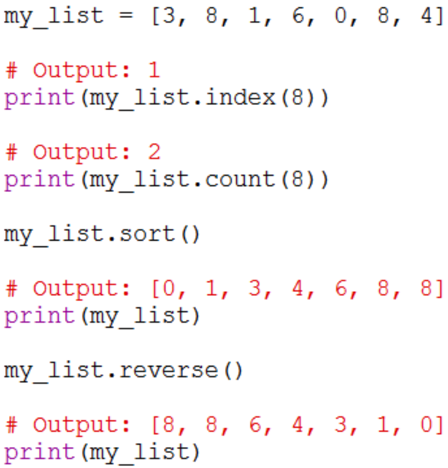
## Delete elements from a list

```
my_list = ['p','r','o','b','l','e','m']
my_list.remove('p')

# Output: ['r', 'o', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'o'
print(my_list.pop(1))

# Output: ['r', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'm'
print(my_list.pop())

# Output: ['r', 'b', 'l', 'e']
print(my_list)

my_list.clear()

# Output: []
print(my_list)
```

## Delete elements from a list

```
mylist = ['p','r','o','b','l','e','m']
mylist[2:3] = []
# mylist
['p', 'r', 'b', 'l', 'e', 'm']
mylist[2:5] = []
# mylist ['p', 'r', 'm']
```

## Python List Methods



```python
my_list = [3, 8, 1, 6, 0, 8, 4]

# Output: 1
print(my_list.index(8))

# Output: 2
print(my_list.count(8))

my_list.sort()

# Output: [0, 1, 3, 4, 6, 8, 8]
print(my_list)

my_list.reverse()

# Output: [8, 8, 6, 4, 3, 1, 0]
print(my_list)
```

## Outline

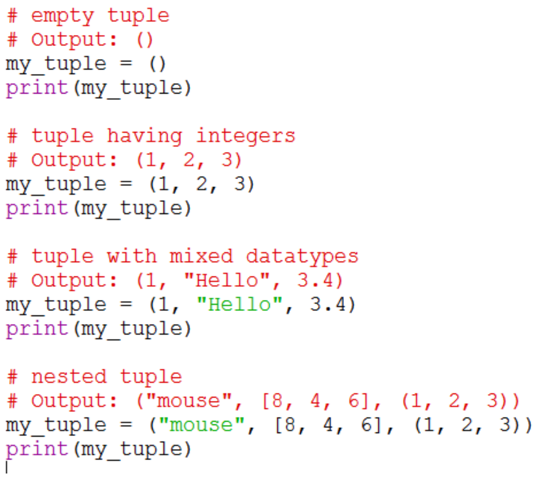
## Python Tuple

A tuple is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed.

**Advantages of Tuple over List**

- Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

## Creating a Tuple



```
# empty tuple
# Output: ()
my_tuple = ()
print(my_tuple)

# tuple having integers
# Output: (1, 2, 3)
my_tuple = (1, 2, 3)
print(my_tuple)

# tuple with mixed datatypes
# Output: (1, "Hello", 3.4)
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

# nested tuple
# Output: ("mouse", [8, 4, 6], (1, 2, 3))
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)
```

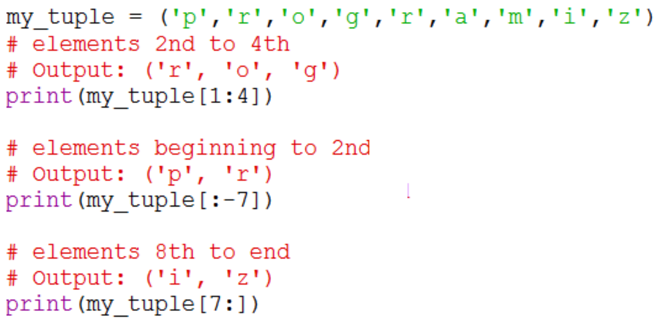
## Creating a tuple with one element

```
# only parentheses is not enough
# Output: <class 'str'>
my_tuple = ("hello")
print(type(my_tuple))

# need a comma at the end
# Output: <class 'tuple'>
my_tuple = ("hello",)
print(type(my_tuple))

# parentheses is optional
# Output: <class 'tuple'>
my_tuple = "hello",
print(type(my_tuple))
```

## Accessing Elements in a Tuple

```
my_tuple = ('p','e','r','m','i','t')
# Output: 'p'
print(my_tuple[0])
# Output: 't'
print(my_tuple[5])

# index must be in range
#print(my_tuple[6])

# index must be an integer
# TypeError: list indices must be integers, not float
#my_tuple[2.0]

# nested tuple
n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# nested index
# Output: 's'
print(n_tuple[0][3])

# nested index
# Output: 4
print(n_tuple[1][1])
```

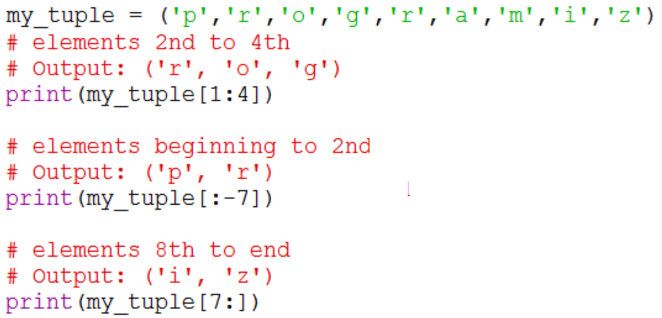
## Negative indexing and slicing in a Tuple



```
my_tuple = ('p','r','o','g','r','a','m','i','z')
# elements 2nd to 4th
# Output: ('r', 'o', 'g')
print(my_tuple[1:4])

# elements beginning to 2nd
# Output: ('p', 'r')
print(my_tuple[:-7])

# elements 8th to end
# Output: ('i', 'z')
print(my_tuple[7:])
```

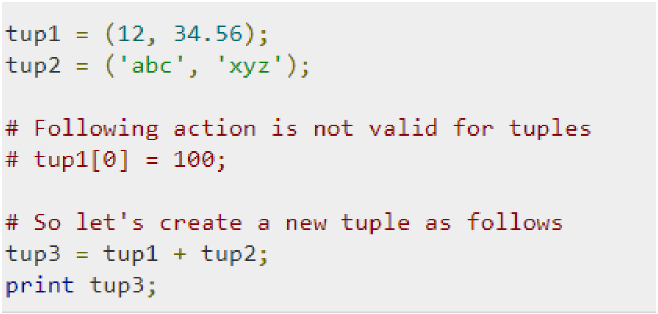
## Changing a Tuple

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples



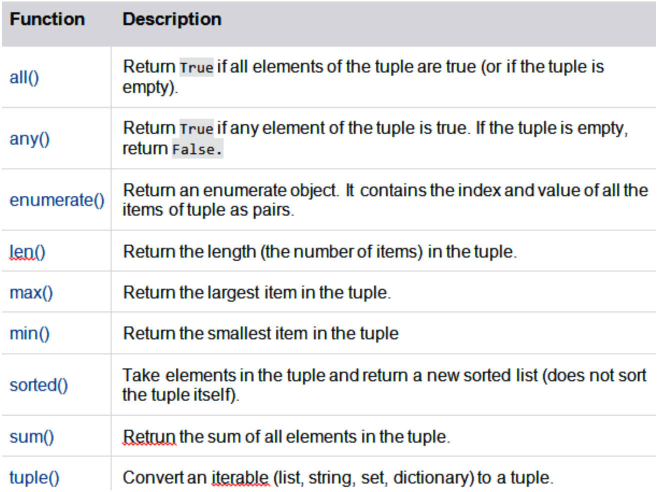
```
my_tuple = ('p','r','o','g','r','a','m','i','z')
# elements 2nd to 4th
# Output: ('r', 'o', 'g')
print(my_tuple[1:4])

# elements beginning to 2nd
# Output: ('p', 'r')
print(my_tuple[:-7])

# elements 8th to end
# Output: ('i', 'z')
print(my_tuple[7:])
```

## Updating Tuple



```
tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');

# Following action is not valid for tuples
# tup1[0] = 100;

# So let's create a new tuple as follows
tup3 = tup1 + tup2;
print tup3;
```

## Built-in Functions with Tuple



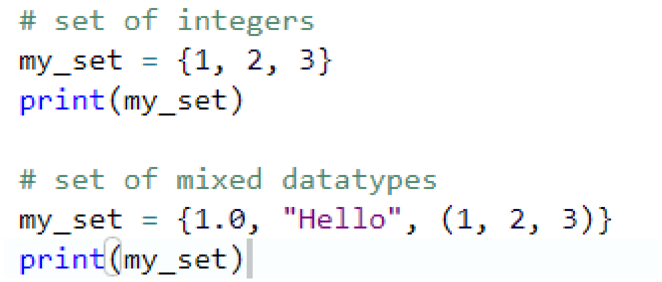
| Function | Description |
|---|---|
| all() | Return `True` if all elements of the tuple are true (or if the tuple is empty). |
| any() | Return `True` if any element of the tuple is true. If the tuple is empty, return `False.` |
| enumerate() | Return an enumerate object. It contains the index and value of all the items of tuple as pairs. |
| len() | Return the length (the number of items) in the tuple. |
| max() | Return the largest item in the tuple. |
| min() | Return the smallest item in the tuple |
| sorted() | Take elements in the tuple and return a new sorted list (does not sort the tuple itself). |
| sum() | Retrun the sum of all elements in the tuple. |
| tuple() | Convert an iterable (list, string, set, dictionary) to a tuple. |

## Outline

## set in Python

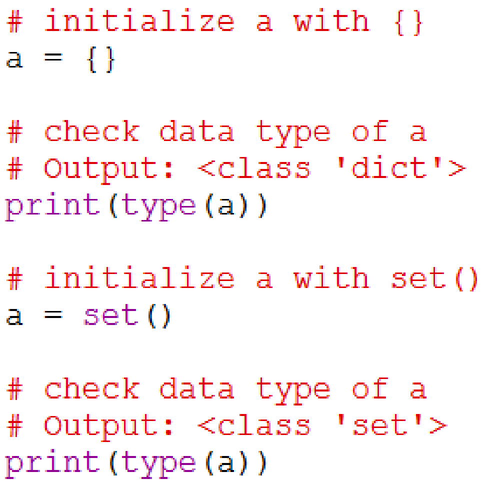
- A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed).
- However, the set itself is mutable. We can add or remove items from it.
- Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

## Creating set in Python



```
# set of integers
my_set = {1, 2, 3}
print(my_set)

# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

## Creating empty set in Python



```python
# initialize a with {}
a = {}

# check data type of a
# Output: <class 'dict'>
print(type(a))

# initialize a with set()
a = set()

# check data type of a
# Output: <class 'set'>
print(type(a))
```

## Add elements in Python Set

```
# initialize my_set
my_set = {1,3}
print(my_set)

# add an element
my_set.add(2)
print(my_set)
# Output: {1, 2, 3}

# add multiple elements
my_set.update([2,3,4])
print(my_set)
# Output: {1, 2, 3, 4}

# add list and set
my_set.update([4,5], {1,6,8})
print(my_set)
# Output: {1, 2, 3, 4, 5, 6, 8}
```

## Remove elements from Python Set

- A particular item can be removed from set using methods, discard() and remove().
- The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition.

## Remove elements from Python Set

```
# initialize my_set
my_set = {1, 3, 4, 5, 6}
print(my_set)

# discard an element
my_set.discard(4)
print(my_set)
# Output: {1, 3, 5, 6}

# remove an element
my_set.remove(6)
print(my_set)
# Output: {1, 3, 5}
```

## Python Set Operations

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use | operator for union
print(A | B)
# Output: {1, 2, 3, 4, 5, 6, 7, 8}

# use union function
A.union(B)
# Output:{1, 2, 3, 4, 5, 6, 7, 8}

# use & operator or intersection() for intersection
print(A & B)
# Output:{1, 2, 3, 4, 5, 6, 7, 8}
A.intersection(B)
# Output:{1, 2, 3, 4, 5, 6, 7, 8}
```

## Outline

## Python Dictionary

- Dictionaryis an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data.

**Creating a dictionary**

```
# empty dictionary
my_dict = {}

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()
my_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

## Python Dictionary

- Access elements from a dictionary

```
mydict = {'name':'Jack', 'age': 26}
# Output: Jack
print(mydict['name'])
# Output: 26
print(mydict.get('age'))
```

- change or add elements in a dictionary

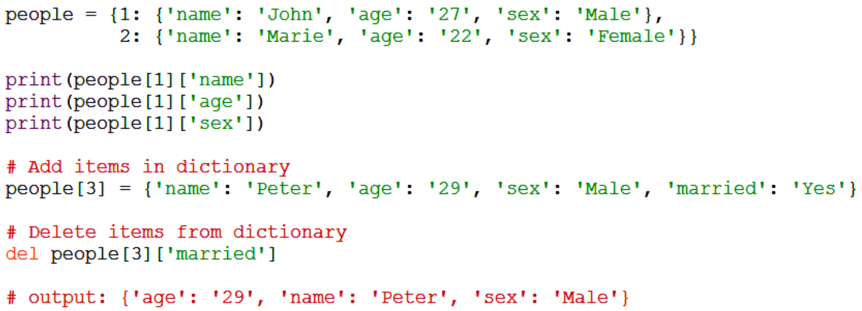
```
mydict = {'name':'Jack', 'age': 26}
# update value
mydict['age'] = 27
# Output: {'age': 27, 'name': 'Jack'}
print(mydict)
# add item
mydict['address'] = 'Downtown'
# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
```

## Python Dictionary

### Delete or remove elements from a dictionary

```
# create a dictionary
squares = {1:1, 2:4, 3:9, 4:16, 5:25}
# remove a particular item
print(squares.pop(4))
# Output: 16
print(squares)
# Output: {1: 1, 2: 4, 3: 9, 5: 25}
# remove an arbitrary item
print(squares.popitem())
# Output: (1, 1)
# delete a particular item
del squares[5]
print(squares)
# Output: {2: 4, 3: 9}
# remove all items
squares.clear()
```

## Python Nested Dictionary



```
people = {1: {'name': 'John', 'age': '27', 'sex': 'Male'},
          2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}}

print(people[1]['name'])
print(people[1]['age'])
print(people[1]['sex'])

# Add items in dictionary
people[3] = {'name': 'Peter', 'age': '29', 'sex': 'Male', 'married': 'Yes'}

# Delete items from dictionary
del people[3]['married']

# output: {'age': '29', 'name': 'Peter', 'sex': 'Male'}
```

## Control Flow in Python

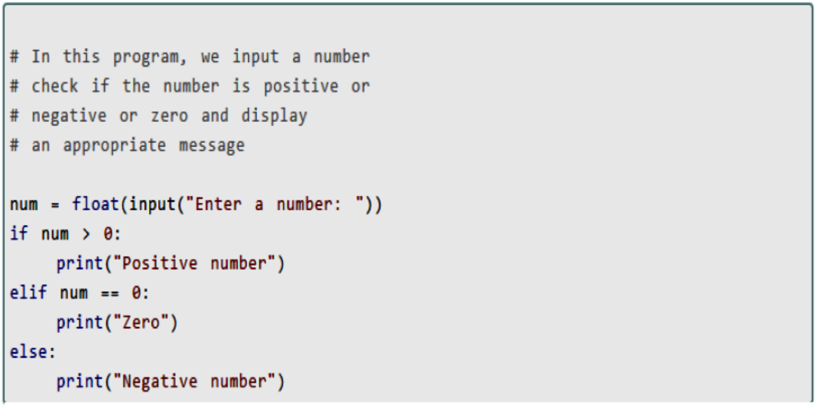
- The if else statement is used in Python for decision making

```
if test expression:
  statement(s)
else:
  statement(s)
```

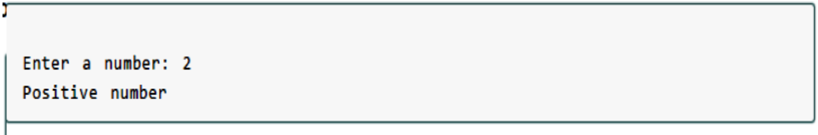
- To test more than one condition following syntax is used

```
if test expression:
  Body of if
elif test expression:
  Body of elif
else:
  Body of else
```

## Control Flow in Python

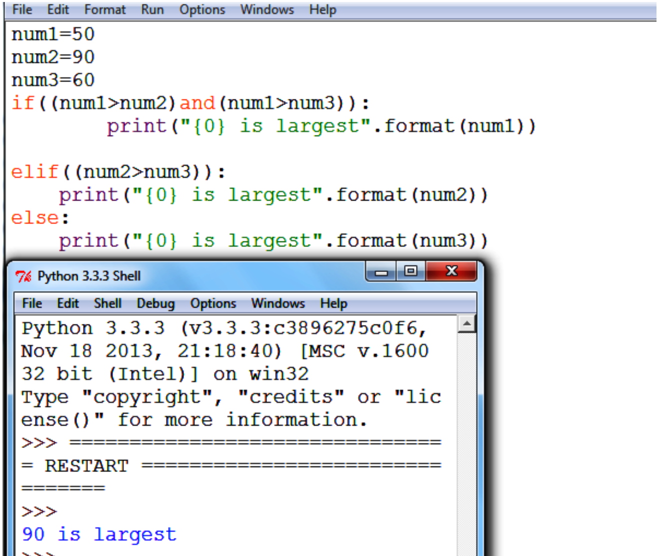


```
# In this program, we input a number
# check if the number is positive or
# negative or zero and display
# an appropriate message

num = float(input("Enter a number: "))
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```



```
Enter a number: 2
Positive number
```

## Control Flow in Python


File Edit Format Run Options Windows Help
Python 3.3.3 Shell
File Edit Shell Debug Options Windows Help

```
num1=50
num2=90
num3=60
if((num1>num2)and(num1>num3)):
        print("{0} is largest".format(num1))

elif((num2>num3)):
    print("{0} is largest".format(num2))
else:
    print("{0} is largest".format(num3))
```

```
Python 3.3.3 (v3.3.3:c3896275c0f6,
Nov 18 2013, 21:18:40) [MSC v.1600
32 bit (Intel)] on win32
Type "copyright", "credits" or "lic
ense()" for more information.
>>> ================================
= RESTART ==========================
=======
>>>
90 is largest
```

## Control Flow in Python

**Things that are False**

- The boolean value False
- The numbers 0 (integer), 0.0 (float) and 0j (complex).
- The empty string "".
- The empty list [], empty dictionary  and empty set().

**Things that are True**

- The boolean value True
- All non-zero numbers.
- Any string containing at least one character.
- A non-empty data structure.

## Outline

## Python Loops

- while loop
- for loop

## Python Loops

### While loop

- `while` **loop**: Executes a group of statements as long as a condition is True.
  - good for *indefinite loops* (repeat an unknown number of times)
- Syntax:

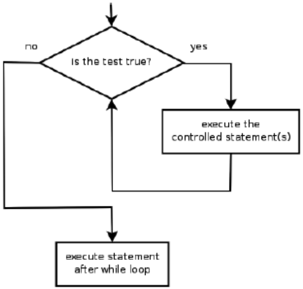
```
while condition:
    statements
```

- Example:

```
number = 1
while number < 200:
    print number,
    number = number * 2
```

  - Output:

```
1 2 4 8 16 32 64 128
```


no
yes
is the test true?
execute the controlled statement(s)
execute statement after while loop

## Python Loops

### The Loop with Else Clause

The optional **else** clause runs only if the loop exits normally (not by break)

```
x = 1

while x < 3 :
    print x
    x = x + 1
else:
    print 'hello'
```

In whileelse.py

```
~: python whileelse.py
1
2
hello
```

Run from the command line

## Python Loops

For loop in python

**Syntax**

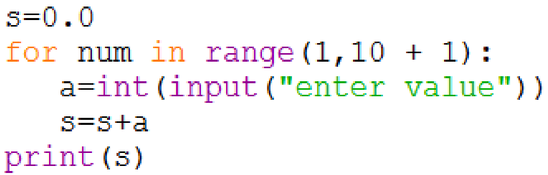
The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversiere.

```
for val in sequence:
  Body of for loop
```

## For loop example

```
# Program to find the sum of all numbers stored in a list

# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
    sum = sum+val

# Output: The sum is 48
print("The sum is", sum)
```

## For loop example



```
s=0.0
for num in range(1,10 + 1):
    a=int(input("enter value"))
    s=s+a
print(s)
```

## Nested Loop

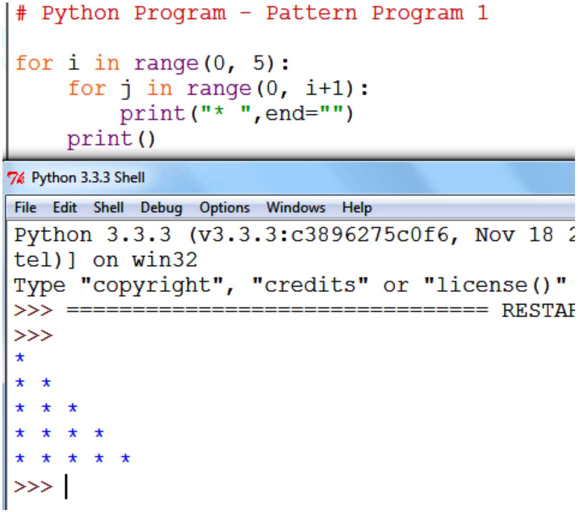
**Nested For Loop**

```
for var1 in sequence:
  for var2 in sequence:
    statements(s)
  statements(s)
```

**Nested while Loop**

```
while expression:
  while expression:
    statement(s)
  statement(s)
```

## Nested loop example

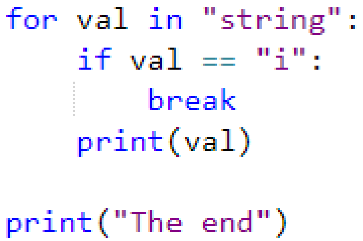


```
# Python Program - Pattern Program 1

for i in range(0, 5):
    for j in range(0, i+1):
        print("* ",end="")
    print()
```

```
Python 3.3.3 (v3.3.3:c3896275c0f6, Nov 18 2
tel)] on win32
Type "copyright", "credits" or "license()"
>>> ================================ RESTAR
>>>
*
* *
* * *
* * * *
* * * * *
>>> 
```

## break statement

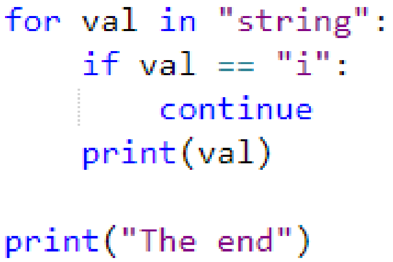
The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.
If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.



```
for val in "string":
    if val == "i":
        break
    print(val)

print("The end")
```

## continue statement

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.



```
for val in "string":
    if val == "i":
        continue
    print(val)

print("The end")
```

## Range function

The range() function returns an immutable sequence object of integers between the given start integer to the stop integer.

range(stop)

- Returns a sequence of numbers starting from 0 to stop - 1
- Returns an empty sequence if stop is negative or 0.

```
print(list(range(0, 5)))
[0, 1, 2, 3, 4]
```

range(start,stop,update)

- print(list(range(10,0,-2)))

```
[10, 8, 6, 4, 2]
```

## Outline

## Exercise 3

- Remove Punctuations from a String
- Sort Words in Alphabetic Order
- Check whether a string is palindrome or not
- Calculate the salary of a camera salesman. His basic salary is 1500, for every camera he will sell he will get 200 and the commission on the months sale is 2 %. The input will be number of cameras sold and total price of the cameras.
- WAP to add all the numbers from 1 to n and n is given by user.
- Python Program to Find the Second Largest Number in a List

## Functions in Python

In Python, function is a group of related statements that perform a specific task.

**Syntax**

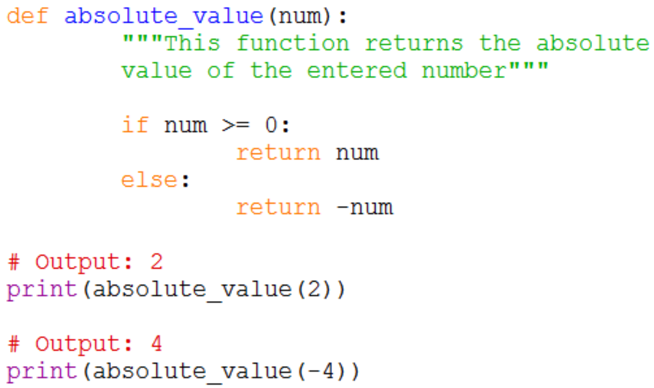
```
def functionname(parameters):
  """docstring"""
    statement(s)
```

- Keyword def marks the start of function header.
- A function name to uniquely identify it. Function naming follows the same rules of writing identifiers in Python.
- Parameters (arguments) through which we pass values to a function. They are optional.
- A colon (:) to mark the end of function header.
- An optional return statement to return a value from the function.

## Python Function

```
def greet(name):
        """This function greets to
        the person passed in as
        parameter"""
        print("Hello, " + name + ". Good morning!")

greet('Paul')
```

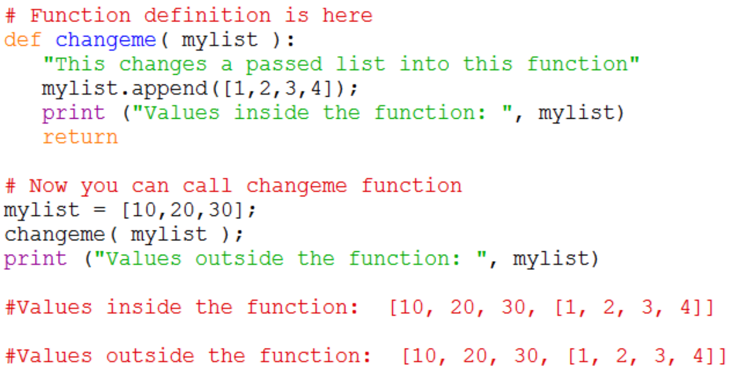
## Python Function



```
def absolute_value(num):
        """This function returns the absolute
        value of the entered number"""

        if num >= 0:
                return num
        else:
                return -num

# Output: 2
print(absolute_value(2))

# Output: 4
print(absolute_value(-4))
```

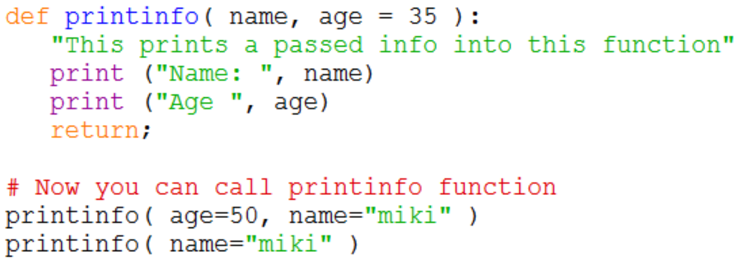
## Pass by reference vs value

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.



```
# Function definition is here
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist.append([1,2,3,4]);
   print ("Values inside the function: ", mylist)
   return

# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print ("Values outside the function: ", mylist)

#Values inside the function:  [10, 20, 30, [1, 2, 3, 4]]

#Values outside the function:  [10, 20, 30, [1, 2, 3, 4]]
```

## Function with default arguments



```
def printinfo( name, age = 35 ):
   "This prints a passed info into this function"
   print ("Name: ", name)
   print ("Age ", age)
   return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```

## Function with Variable-length arguments

```
def printinfo( arg1, *vartuple ):
   "This prints a variable passed arguments"
   print ("Output is: ")
   print (arg1)
   for var in vartuple:
      print (var)
   return;

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

## Scope of Variables

```
total = 0; # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
   # Add both the parameters and return them."
   total = arg1 + arg2; # Here total is local variable.
   print ("Inside the function local total : ", total)
   return total;

# Now you can call sum function
sum( 10, 20 );
print ("Outside the function global total : ", total)


# Inside the function local total :  30
# Outside the function global total :  0
```

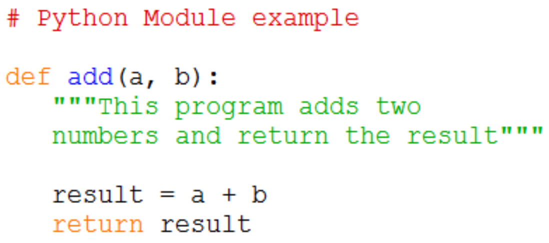
## Recursion in Python

Recursion is the process of defining something in terms of itself.

```
def calc_factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
        return (x * calc_factorial(x-1))

num = 4
print("The factorial of", num, "is", calc_factorial(num))
```

## Python Modules

- Modules refer to a file containing Python statements and definitions.
- A file containing Python code, for e.g.: example.py, is called a module and its module name would be example.
- First create a module example.py



```python
# Python Module example

def add(a, b):
    """This program adds two
    numbers and return the result"""

    result = a + b
    return result
```

## Import modules in Python

- Use import module to import the definition of module
  - import example
  - example.add(4,5.5)

## Python import statement

```python
# to import standard module math

import math
print("The value of pi is", math.pi)

# import module by renaming it

import math as m
print("The value of pi is", m.pi)

# import only pi from math module

from math import pi
print("The value of pi is", pi)

# import all names from the standard module math

from math import *
print("The value of pi is", pi)
```

## Outline

## Excercise 4

- Write a Python function that takes two lists and returns True if they have at least one common member.
- Write a Python program to print a specified list after removing the 0th, 4th and 5th elements.
- Python Program to Count the Number of Vowels Present in a String using Sets.
- Python Program to sum all the Items in a Dictionary.
- Write a Python program to add two matrices.

## File handling in python

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

**File operations**

- Open a file
- Read or write (perform operation)
- Close the file

## File operations

```
# open file from current directory
f = open("test.txt")
# specifying full path
f = open("C:/Python33/README.txt")


f = open("test.txt")       # equivalent to 'r' or 'rt'
f = open("test.txt",'w')   # write in text mode
f = open("img.bmp",'r+b')  # read and write in binary mode

# Close file
f.close()
```

## File operations

- Unlike other languages, the character 'a' does not imply the number 97 until it is encoded using ASCII (or other equivalent encodings).
- Moreover, the default encoding is platform dependent. In windows, it is 'cp1252' but 'utf-8' in Linux.
- Hence, when working with files in text mode, it is highly recommended to specify the encoding type.
  ```
  f = open("test.txt",mode = 'r',encoding = 'utf-8')
  ```
- In case any exception occurs use a try...finally block.
  ```
  try:
     f = open("test.txt",encoding = 'utf-8')
  # perform file operations
  finally:
     f.close()
  ```

## Write to and read from File

```
with open("test.txt",'w',encoding = 'utf-8') as f:
   f.write("my first file\n")
   f.write("This file\n\n")
   f.write("contains three lines\n")
   f.close()

# read files
f = open("test.txt",'r',encoding = 'utf-8')
print(f.read(4))    # read the first 4 data
print(f.read(4))    # read the next 4 data
print(f.read())    # read in the rest till end of file
f.read()  # further reading returns empty sting ''

#output
my f
irst
 file
This file

contains three lines
```

## Write to and read from File

```
with open("test.txt",'w',encoding = 'utf-8') as f:
   f.write("my first file\n")
   f.write("This file\n\n")
   f.write("contains three lines\n")
   f.close()

# read files
f = open("test.txt",'r',encoding = 'utf-8')
print(f.read(4))    # read the first 4 data
print(f.read(4))    # read the next 4 data
print(f.tell())    # get the current file position
f.seek(0)             # bring file cursor to initial position
f.readline()          # read individual lines of a file
print(f.read())    # read in the rest till end of file
f.read()  # further reading returns empty sting ''

#output:
my f
irst
8
This file
```

## Outline

## Excercise 6

- Python Program to Count the Number of Words in a Text File.
- Python Program to Read a String from the User and Append it into a File.

## Python Project

- Create a Python project of a Magic 8 Ball which is a toy used for fortune-telling or seeking advice.
    - Allow the user to input their question.
    - Show a progress message.
    - Create 10/20 responses, and show a random response.
    - Allow the user to ask another question/advice or quit the game.

## For Further Reading I

Learning Python
*David Ascher and Mark Lutz.*
O'Reilly.