# Python Basics

## History of Python

- Python is a programming language created by Guido van Rossum.
  - In December 1989, Van Rossum had been looking for a "'hobby' programming project that would keep [him] occupied during the week around Christmas" as his office was closed.
  - He decided to write an interpreter for a "new scripting language [he] had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers".
  - He attributes choosing the name "Python" to "being in a slightly irreverent mood (and [being] a big fan of Monty Python's Flying Circus)

## Python

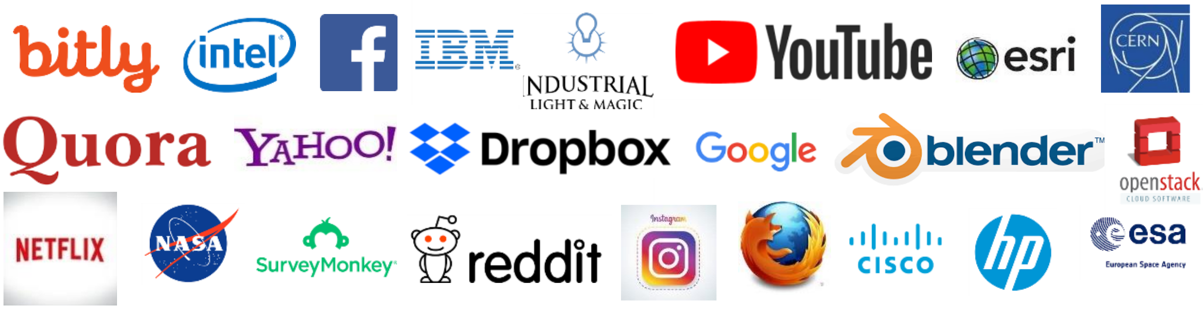
- For Web Development

- For Science

- For Cloud Configura

- For Data Analytics

## Who is using Python?



## What is Python?

- Python is a powerful high-level, object-oriented programming language.
- Python is a general-purpose language.
  - It has wide range of applications from
    - Web development (like: Django and Bottle)
    - Scientific and mathematical computing (Orange, SymPy, NumPy)
    - Desktop graphical user Interfaces (Pygame, Panda3D).

**Features**

- A simple language which is easier to learn
  - Python has a very simple and elegant syntax.
  - It's much easier to read and write Python programs compared to other languages like: C++, Java, C#.
  - Python makes programming fun and allows you to focus on the solution rather than syntax.
- Free and open-source
  - Freely use and distribute Python, even for commercial use.
  - Distribute software's written in it.
  - Make changes to the Python's source code.
  - Python has a large community constantly improving it in each iteration.
- Portability
  - Python programs can be moved from one platform to another, and run without any changes.
  - It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.
- Extensible and Embeddable
  - Pieces of C/C++ or other languages can be combined with Python code.
  - Scripting capabilities of other languages can be used with python.
- A high-level, interpreted language
  - When Python code is run, it automatically converts the code to the language the computer understands.
- Large standard libraries to solve common tasks
  - Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself.
  - Standard libraries in Python are well tested and used by hundreds of people.
- Object-oriented
  - Everything in Python is an object.
  - Object oriented programming (OOP) helps to solve a complex problem intuitively.
  - With OOP, the complex problems can be divided into smaller sets by creating objects.

| **Python Implementation** | **Written in** | **Runs on** |
|---|---|---|
| Cython | THE C PROGRAMMING LANGUAGE | Native Machine |
| Jython | Java | JVM (Java Virtual Machine) |

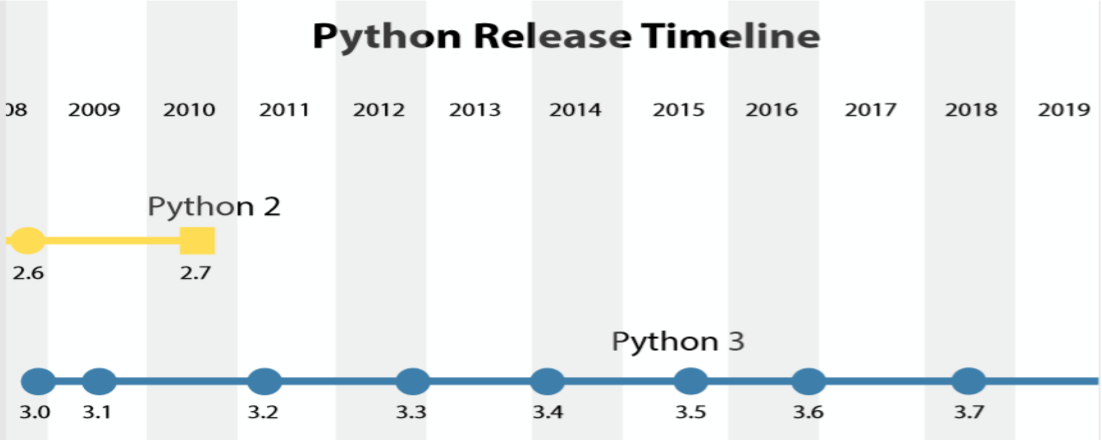
IronPython C# Microsoft .NET

pypy python R

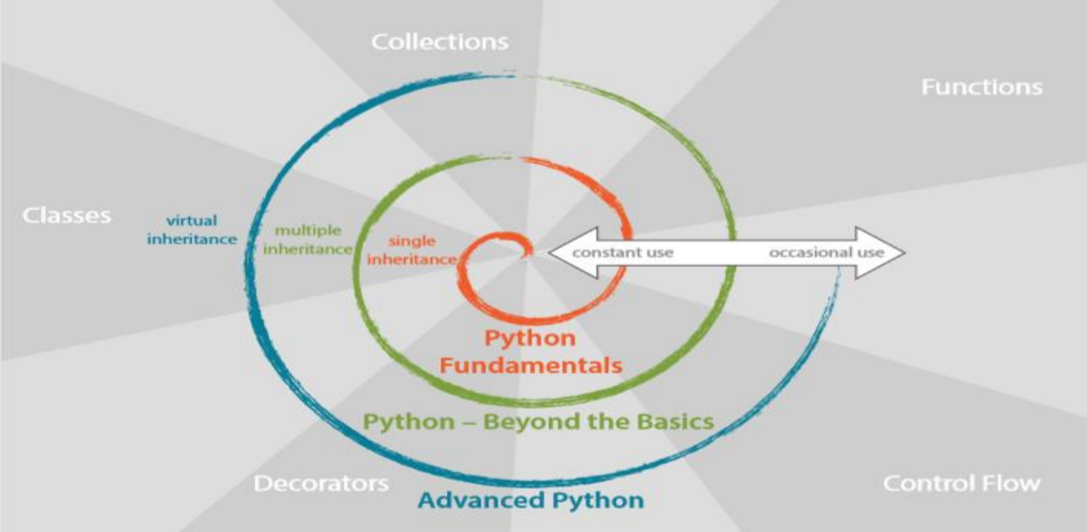
**Native Machines**

**and Others**

**Python Release**



**Python – Basics to Advanced**



**Python Applications**

- **Web Applications**
  - Scalable Web Apps using frameworks and CMS (Content Management System) can be created that are built on Python.
  - Some of the popular platforms for creating Web Apps are: Django, Flask, Pyramid, Plone, Django CMS.
  - Sites like Mozilla, Reddit, Instagram and PBS are written in Python.

- **Scientific and Numeric Computing**
  - There are numerous libraries available in Python for scientific and numeric computing.
  - There are libraries like: SciPy and NumPy that are used in general purpose computing.
  - There are specific libraries like: EarthPy for earth science, AstroPy for Astronomy and so on.
  - The language is heavily used in machine learning, data mining and deep learning.
- **Creating software Prototypes**
  - Python is slow compared to compiled languages like C++ and Java.
  - It might not be a good choice if resources are limited and efficiency is a must.
  - However, Python is a great language for creating prototypes.
- **Good Language to Teach Programming**
  - Python is used by many companies to teach programming to kids and newbies.
  - It is a good language with a lot of features and capabilities.
  - It's one of the easiest languages to learn because of its simple easy-to-use syntax.

**Windows Installation**

- Go to https://www.python.org/downloads/
- Download Python 3.7.0. or Python 3.6.6.

**Ubuntu Installation**

- Open terminal and run command
  - **sudo add-apt-repository ppa:jonathonf/python-3.6**
- Check updates and install Python 3.6
  - **sudo apt-get update**
  - **sudo apt-get install python3.6**
- To make python3 use the new installed python 3.6 instead of the default 3.5 release, run following 2 commands:
  - **sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.5 1**
  - **sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.6 2**

**IDE**

- gedit
- sublime
- spyder
- pycharm

**Python Interpreter - Python 3**

- After installation, the python interpreter lives in the installed directory.
  - **It is /usr/local/bin/pythonX.X in Linux/Unix.**
  - **It is C:\PythonXX in Windows, where the 'X' denotes the version number.**
- To invoke it from the shell or the command prompt add this location in the search path.
- Search path is a list of directories (locations) where the operating system searches for executables.
- Example
  - In Windows command prompt, path=%path%;c:\python33 (python33 means version 3.3, it might be different in your case) to add the location to path for that particular session.

**Use python**

- Now there are various ways to start Python.
- **Immediate mode**
  - Typing python in the command line will invoke the interpreter in immediate mode.
  - Directly type in Python expressions and press enter to get the output.
  - >>>
    - is the Python prompt.
    - It tells us that the interpreter is ready for our input.
  - To exit this mode type exit() or quit() and press enter.
- **Script mode**
  - This mode is used to execute Python program written in a file.
  - Such a file is called a script.
    - Scripts can be saved to disk for future use.
    - Python scripts have the extension .py.
  - To execute the file in script mode simply write python <filename>.py at the command prompt.
- **Integrated Development Environment (IDE)**
  - Use any text editing software to write a Python script file.
  - Save it with the .py extension.
- IDLE is a graphical user interface (GUI) that can be installed along with the Python programming language and is available from the official website.

**Statements- Single and Multi-line**

- Instructions that a Python interpreter can execute are called statements.
- The end of a statement is marked by a newline character.

- A statement can be extended over multiple lines with the line continuation character (\).
- Line continuation is implied inside parentheses ( ), brackets [ ] and braces { }.

**Lines and Indentation**

- No braces to indicate blocks of code for class and function definitions or flow control.
- Blocks of code are denoted by line indentation, which is rigidly enforced.
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.
- Statements contained within the [], {} or () brackets do not need to use the line continuation character.

**Comments in Python**

- A hash sign (#) that is not inside a string literal begins a comment.
- All characters after the # and up to the physical line end are part of the comment and the Python interpreter ignores them.
  - # This is a comment

**Using Blank Lines**

- A line containing whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.
- In an interactive interpreter session, an empty physical line should be entered to terminate a multiline statement

**Multiple Statements on a Single Line**

- The semicolon ( ; ) allows multiple statements on the single line given that neither statement starts a new code block
  - x =4; y = 6; s = x+y; print(s) #(correct)
  - x=3; y=5; if x<y: #(wrong)

**Multiple Statement Groups as Suites**

- A group of individual statements, which make a single code block are called suites in Python.
- Compound or complex statements, such as if, while, def, and class require a header line and a suite.
  - if expression :
  -                 suite
  - elif expression :
  -                  suite
  - else :
  -                suite

**Keywords**

- Keywords are the reserved words in Python.
- Keywords cannot be used as variable name, function name or any other identifier.
- They are used to define the syntax and structure of the Python language.
- In Python, keywords are case sensitive.
- There are 33 keywords in Python 3.3.
- All the keywords except True, False and None are in lowercase and they must be written as it is.

| False | class | finally | is | return | assert | Else |
|---|---|---|---|---|---|---|
| None | continue | for | lambda | try | break | except |
| True | Def | from | nonlocal | while | | |
| and | Del | global | not | with | | |
| as | Elif | if | or | yield | | |

**Identifiers**

- Identifier is the name given to entities like class, functions, variables etc. in Python.
- It helps differentiating one entity from another.

**Rules for identifiers**

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
  - Examples - myClass, var_1 and print_this_to_screen, all are valid example.
- An identifier cannot start with a digit.
  - Example -1variable is invalid
  - - variable1 is perfectly fine.
- Keywords cannot be used as identifiers.
- Special symbols like !, @, #, $, % etc. cannot be used.
- Identifier can be of any length.

**Variables**

- A variable is a way of referring to a memory location used by a computer program.
- A variable is a symbolic name for this physical location.
- This memory location contains values, like numbers, text or more complicated types.
- Python is a type inferred language, it can automatically infer the type of the variable.
- A variable is created the moment it is first assigned a value.
- Variables do not need to be declared with any particular type and can even change type after they have been set.
- A single value can be assigned to several variables simultaneously.

**Constant**

- A constant is a type of variable whose value cannot be changed.
- They are containers that hold information which cannot be changed later.

**Literals**

- Literal is a raw data given in a variable or constant.
- Python allows
  - Numeric Literals
  - String literals
  - Boolean literals
  - Special literals - None
- **Numeric Literals**
  - Numeric Literals are immutable (unchangeable).
  - Numeric literals can belong to 3 different numerical types
    - Integer
    - Float
    - Complex
- **String Literals**
  - A string literal is a sequence of characters surrounded by quotes.
  - Single, double or triple quotes are used for a string.
  - A character literal is a single character surrounded by single or double quotes.
- **Boolean Literals**
  - A Boolean literal can have any of the two values
    - True
    - False
- **Special Literals**
  - Python contains one special literal i.e. None.
  - It is used to specify to that field that is not created.

**Data Types**

- Numbers
- List
- Tuple
- Strings
- Set
- Dictionary
- **Numbers**
  - Integers – int
    - Integers can be of any length, it is only limited by the memory available.

  - Floating point numbers – float
    - A floating-point number is accurate up to 15 decimal places.
    - Integer and floating points are separated by decimal points. 1 is integer, 1.0 is floating point number.
  - Complex numbers - complex
    - Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part.

- **Strings**
  - String is sequence of Unicode characters.
  - Single quotes or double quotes can be used to represent strings.
  - Multi-line strings can be denoted using triple quotes, ''' or """.
  - A string in Python consists of a series or sequence of characters - letters, numbers, and special characters.
  - Strings can be indexed - often synonymously called subscripted as well.
  - Similar to C, the first character of a string has the index 0.
- **Set**
  - Set is an unordered collection of unique items.
  - Set is defined by values separated by comma inside braces { }.
  - Items in a set are not ordered.
- **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.
  - Declaring a list is pretty straight forward.
    - Items separated by commas are enclosed within brackets [ ].
- **Tuple**
  - Tuple is an ordered sequence of items same as list.
  - The only difference is that tuples are immutable.
  - Tuples once created cannot be modified.
  - Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically.
  - It is defined within parentheses () where items are separated by commas.
- **Dictionary**
  - Dictionary is an unordered collection of key-value pairs.
  - It is generally used when a huge amount of data is present.
  - Dictionaries are optimized for retrieving data.
  - The key must be known to retrieve the value.

- Dictionaries are defined within braces {} with each item being a pair in the form key:value.
- Key and value can be of any type.

**Casting or Conversion**

- In Implicit type conversion, Python automatically converts one data type to another data type.
- Python promotes conversion of lower datatype (integer) to higher data type (float) to avoid data loss.
- Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.
- Casting in python is therefore done using constructor functions:
    - int()
    - float()
    - str()
    - int()
        - Constructs an integer number from
            - An integer literal
            - A float literal (by rounding down to the previous whole number) literal
            - String literal (providing the string represents a whole number)
    - float()
        - Constructs a float number from
            - An integer literal
            - A float literal
            - A string literal (providing the string represents a float or an integer)
    - str()
        - Constructs a string from a wide variety of data types including
            - Strings
            - Integer literals
            - Float literals
- **Key-Points**
    - Type Conversion is the conversion of object from one data type to another data type.
    - Implicit Type Conversion is automatically performed by the Python interpreter.
    - Python avoids the loss of data in Implicit Type Conversion.
    - Explicit Type Conversion is also called Type Casting, the data types of object are converted using predefined function by user.
    - In Type Casting loss of data may occur as the object is enforced to a specific data type

**Operators**

₪ Arithmetic operators

₪ Assignment operators

₪ Comparison operators

₪ Logical operators

₪ Identity operators

₪ Membership operators

₪ Bitwise operators

**Arithmetic Operators**

₪ + Addition x + y

₪ - Subtraction x - y

₪ * Multiplication x * y

₪ / Division x / y

₪ % Modulus x % y

₪ ** Exponentiation x ** y

₪ // Floor division x // y

**Assignment Operators**

₪ = x = 5 x = 5

₪ += x += 5x = x + 5

₪ -= x -= 5 x = x – 5

₪ *= x *= 5 x = x * 5

₪ /= x /= 5 x = x / 5

₪ %= x %= 5 x = x % 5

₪ //= x //= 5 x = x // 5

₪ **=x **= 5 x = x ** 5

₪ &= x &= 5x = x & 5

₪ |= x |= 5 x = x | 5

₪ ^= x ^= 5x = x ^ 5

₪ >>= x >>= 5 x = x >> 5

₪ <<= x <<= 5 x = x << 5

**Comparison Operators**

₪ Comparison operators are used to compare values.

₪ It either returns True or False according to the condition.

∞ > Greater that - True if left operand is greater than the right x > y

∞ < Less that - True if left operand is less than the right x < y

∞ == Equal to - True if both operands are equal x == y

∞ != Not equal to - True if operands are not equal x != y

∞ >= Greater than or equal to - True if left operand is greater than or equal to the right x >= y

- ∞ <= Less than or equal to - True if left operand is less than or equal to the rightx <= y

**Logical Operators**

- ₪ and True if both the operands are true x and y
- ₪ or True if either of the operands is true x or y
- ₪ not True if operand is false (complements the operand) not x

**Bitwise Operators**

- ₪ Bitwise operators act on operands as if they were string of binary digits.
  - ∞ &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)

**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.
- ₪ Two variables that are equal does not imply that they are identical.
  - ∞ is True if the operands are identical (refer to the same object) x is True
  - ∞ is not True if the operands are not identical (do not refer to the same object) x is not True

**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.
  - ∞ in True if value/variable is found in the sequence 5 in x
  - ∞ not in True if value/variable is not found in the sequence 5 not in x

**Input – Output**

- ₪ To take the input from the user input() is used.
- ₪ The syntax for input() is
  - ∞ input([prompt])
    - δ where prompt is the string we wish to display on the screen.

**Import**

- ₪ A module is a file containing Python definitions and statements.
- ₪ Python modules have a filename and end with the extension .py.
- ₪ When the program grows bigger, it is a good idea to break it into different modules.
- ₪ When people have already written a few modules it is better to utilize the module.

- Definitions inside a module can be imported to another module or the interactive interpreter in Python.
- Use the import keyword to do this.

## Working with Data types

### Strings

- Slice Operator – [] with subscript 0 for beginning of the string and -1 for the end.
- The plus ( + ) sign is the string concatenation operator
- The asterisk ( * ) is the repetition operator.
- The [x : y] allows the string to be displayed from 'x' position till 'y' number of characters are displayed.
- Characters inside the string cannot be changed.
- strip() removes any whitespace from the beginning or the end.
    - string.strip()
- len() method returns the length of a string.
    - len(string)
- lower() method returns the string in lower case.
- upper() method returns the string in upper case.
    - string.lower() or string.upper()
- replace() method replaces a string with another string.
    - string.replace("character to be replaced", "character that replaces")
- split() method splits the string into substrings if it finds instances of the separator.
    - string.split("sep")
- capitalize() converts the first character to upper case.
    - string.capitalize()
- count() returns the number of times a specified value occurs in a string
    - string.count("Pattern")
- find() searches the string for a specified value and returns the position of where it was found first
    - string.find("pattern")
- islower() returns True if all characters in the string are lower case
- isupper() returns True if all characters in the string are upper case
    - string.islower() and string.isupper()
- isalnum() function returns True if all characters in the string are alphanumeric
    - string.isalnum()
- isalpha() function returns True if all characters in the string are in the alphabet
    - string.isalpha()
- isdecimal() function returns True if all characters in the string are decimals

  - string.isdecimal()
- isdigit() function returns True if all characters in the string are digits
  - string.isdigit()
- isidentifier() function returns True if the string is an identifier
  - string.isidentifier()
- isnumeric() function returns True if all characters in the string are numeric
  - string.isnumeric()
- isprintable() function returns True if all characters in the string are printable
  - string.isprintable()
- isspace() function returns True if all characters in the string are whitespaces
  - string.isspace()
- istitle() function returns True if the string follows the rules of a title
  - string.isspace()
- join() function joins the elements of an iterable to the end of the string
  - string.join(iterable)
- partition() function returns a tuple where the string is parted into three parts
  - string.partition(value)
- split() function splits the string at the specified separator, and returns a list
  - string.split(separator, max)

**Working on Tuples**

- tuple() method to make a tuple.
  - Tuplename = tuple((“value1", “value2", “value3"))
- len() method to find the length of a tuple.
- Slice Operator – [] with subscript 0 for beginning of the string and -1 for the end.
- The [x : y] allows the tuple values to be displayed from ‘x’ position till ‘y’ number of total tuples are displayed.
- Tuple values once created cannot be modified.

**Working on Sets**

- set() constructor make a set
  - Setname = set((“value1”, “value2”,…))
- add() method add an item
  - Setname.add(“value”)
- remove() method remove an item
  - Setname.remove(“value”)
- len() method return the number of items
  - len(Setname)

**Working on Lists**

- list() constructor to make a List
  - Listname = list((“value”, “value”,…….))
- append() append an item
  - Listname.append(“newvalue”)
- remove() remove an item
  - Listname.remove(“value”)
- len() returns the number of items in a list
  - len(Listname)
- clear() removes all the elements from the list
  - Listname.clear()
- copy() returns a copy of the list
  - Listname.copy()
- count() returns the number of elements with the specified value
  - Listname.count(“name”)
- extend() add the elements of a list (or any iterable), to the end of the current list
  - Listname.extend(list)
- index() returns the index of the first element with the specified value
  - list.index(“value”)
- insert() adds an element at the specified position
  - list.insert(position, “element”)
- pop() removes the element at the specified position
  - list.pop(position)
- remove() removes the first item with the specified value
  - list.remove(“element”)
- reverse() reverses the order of the list
  - list.reverse()
- sort() sorts the list
  - list.sort()

**Working on Dictionary**

- dict() constructor make a dictionary
  - Dictname = dict(“key”=“value", “key=“value", “key”=“value")
- Adding an item to the dictionary is done by using a new index key and assigning a value to it
  - Dictname[“key"] = “value"
- del() function remove a dictionary item
  - del(Dictname[“key"])
- len() function returns the size of the dictionary

- len(Dictname)

**Programs**

| | | |
|---|---|---|
| x = 15 | x = 60 | x = int(1) |
| y = 4 | y = 20 | y = int(2.8) |
| print("On Integer's") | print('"',x, '" & "', y, '"is ', x&y) | z = int("3") |
| print('x + y =',x+y) | print('"',x, '" \| "', y, '"is ', x\|y) | print("x =", type(x), x) |
| print('x - y =',x-y) | print('"',x, '" ^ "', y, '"is ', x^y) | print("y =",type(y), y) |
| print('x * y =',x*y) | print('"',x, '" >> 2 is ', x>>2) | print("z =",type(z), z) |
| print('x / y =',x/y) | print('"',x, '" << 2 is ', x<<2) | x = float(1) |
| print('x // y =',x//y) | print('"~',x, '" is ',~x) | y = float(2.8) |
| print('x ** y =',x**y) | x = (1 == True) | z = float("3") |
| x = 15.5 | y = (1 == False) | w = float("4.2") |
| y = 4.9 | a = True + 4 | print("x =",type(x), x) |
| print("On Float") | b = False + 10 | print("y =",type(y), y) |
| print('x + y =',x+y) | print("x is", x) | print("z =",type(z), z) |
| x = str("s1") | print("y is", y) | print("w =",type(w), w) |
| y = str(2) | print("a:", a) | friends = { |
| z = str(3.0) | print("b:", b) | 'tom' : '111-222-333', |
| print("x =",type(x), x) | x = 10 | 'jerry' : '666-33-111' |
| print("y =",type(y), y) | y = 12 | } |
| print("z =",type(z), z) | print( x,' > ', y, ' is',x>y) | print(friends['tom']) |
| d = {1:'value','key':2} | print('x < y  is',x<y) | friends['bob'] = '888-999-666' |
| print(type(d)) | print('x == y is',x==y) | print(friends) |
| print("d[1] = ", d[1]); | print('x != y is',x!=y) | thisdict = dict(apple="green", banana="yellow", cherry="red") |
| print("d['key'] = ", d['key']); | print('x >= y is',x>=y) | print(thisdict) |
| # Generates error | print('x <= y is',x<=y) | thisdict["damson"] = "purple" |
| #print("d[2] = ", d[2]); | x1 = 5 | print(thisdict) |
| num_int = 123 | y1 = 5 | del(thisdict["banana"]) |
| num_str = "4506" | x2 = 'Hello' | print(thisdict) |

| | | |
|---|---|---|
| print("Data type of num_int:",type(num_int)) | y2 = 'Hello' | print(len(thisdict)) |
| print("Data type of num_str before Type Casting:",type(num_str)) | x3 = [1,2,3] | x = True |
| num_str = int(num_str) | y3 = [1,2,3] | y = False |
| num_sum = num_int + num_str | print(x1 is not y1) | print('"',x, '" and "', y, '" is',x and y) |
| print("Sum of num_int and num_str:",num_sum) | print(x2 is y2) | print('"', x, '" or "', y,'" is',x or y) |
| print("Data type of the sum:",type(num_sum)) | print(x3 is y3) | print('not "',x,'" is',not x) |
| x = ["apple", "banana"] | x = ["apple", "banana"] | x = 'Hello world' |
| print("banana" in x) | y = ["apple", "banana"] | y = {1:'a',2:'b'} |
| print("pineapple" not in x) | z = x | print('H' in x) |
| print("banana" not in x) | print(x is z) | print('hello' not in x) |
| a = 5 | print(x is y) | print(1 in y) |
| print(a, "is of type", type(a)) | print(x == y) | print('a' in y) |
| a = 2.0 | num_int = 123 | print(1,2,3,4) |
| print(a, "is of type", type(a)) | num_flo = 1.23 | print(1,2,3,4,sep='*') |
| a = 1+2j | num_new = num_int + num_flo | print(1,2,3,4, sep='"') |
| print(a, "is complex number?", isinstance(1+2j,complex)) | print("datatype of num_int:",type(num_int)) | print(1,2,3,4,sep='!') |
| a = {5,2,3,1,4} | print("datatype of num_flo:",type(num_flo)) | print(1,2,3,4,sep='#',end='&') |
| print("a = ", a) | print("Value of num_new:",num_new) | print(1,2,3,4,sep='$',end='-') |
| print(type(a)) | print("datatype of num_new:",type(num_new)) | strings = "This is Python" |
| thisset = {"apple", "banana", "cherry"} | num_int = 123 | char = "C" |

| | | |
|---|---|---|
| print(thisset) | num_str = "456" | multiline_str = """This is a multiline string with more than one line code.""" |
| print(type(thisset)) | print("Data type of num_int:",type(num_int)) | unicode = u"\u00dcnic\u00f6de" |
| thisset = set(("apple", "banana", "cherry")) # note the double round-brackets | print("Data type of num_str:",type(num_str)) | raw_str = r"raw \n string" |
| print(thisset) | print(num_int+num_str) | print(strings) |
| thisset.add("damson") | num = input('Enter a number: ') | print(char) |
| print(thisset) | print(num, type(num)) | print(multiline_str) |
| thisset.remove("banana") | num = int(input('Enter a number: ')) | print(unicode) |
| print(thisset) | print(num, type(num)) | print(raw_str) |
| print(len(thisset)) | num = float(input('Enter a number: ')) | s = "A string consists of characters" |
| a = "   hello, world!   " | print(num, type(num)) | print(s[0]) |
| print(a) | thislist = list(("apple", "banana", "cherry")) # note the double round-brackets | print(s[len(s)-1]) |
| print("Length of the string is ",len(a)) | print(thislist) | print(s[-1], s[-2]) |
| print(a.strip()) | thislist.append("strawberry") | print(s[2:5]) |
| print("Length of the string is ",len(a)) | print(thislist) | print(s[5:]) |
| print("Lower case - ",a.lower()) | thislist.remove("banana") | print(s[:9]) |
| print("Upper case - ",a.upper()) | print(thislist) | print(s*2) |
| print("Replacement is ",a.replace("H", "XX")) | fruits = ['apple', 'banana', 'cherry', 'orange'] | print(s+" and numbers") |
| print("After Splitting", a.split(",")) | print(fruits.copy()) | s=str(input("Enter a string")) |
| a = "hello, world!   " | print(fruits.count('banana')) | t=s[0] |

| | | |
|---|---|---|
| print("After Capitalizing", a.capitalize()) | fruits = ['apple', 'banana', 'cherry'] | r=s[-1] |
| txt = "I love apples, apple are my favorite fruit, apple pie is my favorite desert" | cars = ['Ford', 'BMW', 'Volvo'] | print(t,r) |
| print("apple in ", txt," is ", txt.count("apple")) | fruits.extend(cars) | s=s.replace(s[-1],t) |
| txt = "Hello, welcome to my world." | print(fruits) | s=s.replace(s[0],r) |
| print("Finding 'welcome' in ",txt,"is at", txt.find("welcome")) | print(fruits.index("BMW")) | print(s) |
| print (a*2) | fruits.insert(1, "orange") | t = (5,'program', 1+3j) |
| print (a + "What's up") | print(fruits) | # t[1] = 'program' |
| txt = "hello, my name is Peter, I am 26 years old" | values = input("Input some comma seprated numbers : ") | print("t[1] = ", t[1]) |
| x = txt.split(", ") | list = values.split(",") | # t[0:3] = (5, 'program', (1+3j)) |
| print(x) | tuple = tuple(list) | print("t[0:3] = ", t[0:3]) |
| myTuple = ("John", "Peter", "Vicky") | print('List : ',list) | # Generates error |
| x = "#".join(myTuple) | print('Tuple : ',tuple) | # Tuples are immutable |
| print(x) | | #t[0] = 10 |
| a = 0b1010  # Binary Literals | | |
| b = 100  # Decimal Literal | | |
| c = 0o310  # Octal Literal | | |
| d = 0x12c  # Hexadecimal Literal | | |
| float_1 = 10.5# Float Literal | | |
| float_2 = 1.5e2 | | |
| x = 3.14j # Complex Literal | | |
| print(a, b, c, d) | | |
| print(float_1, float_2) | | |