Unit 2
Introduction to Python

Prepared By:
Tanvi Patel
Asst. Prof. (CSE)

Contents

History of Python

- Python was conceptualized in the late 1980s and its implementation began in 1989 by Guido van Rossum who then worked in a project called “Amoeba” at the CWI Netherlands, Amsterdam.
- The inspiration for the name came from BBC’s TV Show – ‘Monty Python’s Flying Circus’.
- By nature, it is a high-level programming language that allows for the creation of both simple as well as complex operations.
- Along with this Python comes inbuilt with a wide array of modules as well as libraries which allows it to support many different programming languages like Java, C, C++, and JSON.

Introduction to Python

- Python is an easy to learn, powerful programming language.
- Python is programming language as well as scripting language.
- Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.
- Python is an interpreted and object-oriented programming language.
- Due to its elegance and simplicity, top technology organizations like Dropbox, Google, Quora, Mozilla, Hewlett-Packard, Qualcomm, IBM, and Cisco have implemented Python.

Introduction to Python (Cont.)

- In 1994, Python 1.0 was released with new features like lambda, map, filter, and reduce.
- Python 2.0 added new features such as list comprehensions, garbage collection systems.
- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify the fundamental flow of the language.
- *ABC programming language* is said to be the predecessor of Python language, which was capable of Exception Handling and interfacing with the Amoeba Operating System.
- The following programming languages influence Python:
  - ABC language.
  - Modula-3

Introduction to Python (Cont.)

- **It is used for:**

1. web development (server-side),
2. software development,
3. mathematics,
4. system scripting.

- **What can Python do?**

1. Python can be used on a server to create web applications.
2. Python can be used alongside software to create workflows.
3. Python can connect to database systems. It can also read and modify files.
4. Python can be used to handle big data and perform complex mathematics.
5. Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

1. Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
2. Python has a simple syntax similar to the English language.
3. Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
4. Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
5. Python can be treated in a procedural way, an object-orientated way or a functional way.

Why Python? (Cont.)

Python Syntax compared to other programming languages

1. Python was designed for readability, and has some similarities to the English language with influence from mathematics.
2. Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
3. Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Versions of Python

What is Python 2?

- Python 2.0 was released on 16th Oct, 2000 with a new technical specification called the Python Enhancement Proposal (PEP) along with some additional features such as list comprehension, increased Unicode support, cycle-detecting garbage collection system, and more. It continued to develop over time with updated releases that further added functionalities to the programming language with the version 2.7.

Versions of Python (Cont.)

What is Python 3?

- Python 3 is the next generation of the programming language released in Dec 2008 along with several new enhancements and features, including some deprecated features. The version is completely different from its predecessors and is currently in development to replace the version 2.7.
- The version 3 was released to address the security problems and design flaws in the previous releases, thereby removing redundancy.
- Few of the major changes in the Python 3 include change of print statement into a built-in function, improved integer division, better Unicode support, and more.

Versions of Python (Cont.)

Python 2	Python3
Syntax is comparatively difficult to understand.	Syntax is simpler and easily understandable
ASCII string type is used by default to store strings.	Unicode is the implicit string type by default.
The print syntax is treated as a statement rather than a function which requires text to wrapped in parenthesis. print “hello world”	The print is explicitly treated as a function and replaced by print() function in Python 3 which requires an extra pair of parenthesis. print(“hello world”)
It simply returns an integer to the nearest whole number when dividing two integers.	It makes integer division more intuitive by using true division for integers and floats.
In Python 2, the xrange() is used for iterations.	The new Range() function introduced to perform iterations.
Many recent developers are creating libraries which you can only use with Python 3.	Many older libraries created for Python 2 is not forward-compatible.

Versions of Python (Cont.)

Python 1.0 - Jan 1994
Python 1.5 - 31 Dec 1997
Python 1.6 - 05 Sep 2000
Python 2.0 - 16 Oct 2000
Python 2.1 - 17 Apr 2001
Python 2.2 - 21 Dec 2001
Python 2.3 - 29 Jul 2003
Python 2.4 - 30 Nov 2004
Python 2.5 - 19 Sep 2006
Python 2.5.6 - 26 May 2011
Python 2.6.9 - 29 Oct 2013
Python 2.7.10 - 23 May 2015 -
Python 2.7.13 - 17 Dec 2016 -
Python 3.5.3 - 17 Jan 2017 -
Python 3.6.10 - 18 Dec 2019 -
python 3.8.2 - 24 Feb 2020
Python 3.9.0 - Latest

Features of Python

1. Easy to Code: Python is a very developer-friendly language which means that anyone and everyone can learn to code it in a couple of hours or days. As compared to other object-oriented programming languages like Java, C++, and C#, Python is one of the easiest to learn.

2. Open Source and Free: Python is an open-source programming language which means that anyone can create and contribute to its development. Python has an online forum where thousands of coders gather daily to improve this language further. Along with this Python is free to download and use in any operating system, be it Windows, Mac or Linux.

Features of Python (Cont.)

3. Support for GUI: GUI or Graphical User Interface is one of the key aspects of any programming language because it has the ability to add flair to code and make the results more visual. Python has support for a wide array of GUIs which can easily be imported to the interpreter, thus making this one of the most favorite languages for developers.

4. **Object-Oriented Approach**: One of the key aspects of Python is its object-oriented approach. This basically means that Python recognizes the concept of class and object encapsulation thus allowing programs to be efficient in the long run.

Features of Python (Cont.)

5. High-Level Language: Python has been designed to be a high-level programming language, which means that when you code in Python you don't need to be aware of the coding structure, architecture as well as memory management.

6. Integrated by Nature: Python is an integrated language by nature. This means that the python interpreter executes codes one line at a time. Unlike other object-oriented programming languages, we don't need to compile Python code thus making the debugging process much easier and efficient. Another advantage of this is, that upon execution the Python code is immediately converted into an intermediate form also known as byte-code which makes it easier to execute and also saves runtime in the long run.

Features of Python (Cont.)

7. Highly Portable: Suppose you are running Python on Windows and you need to shift the same to either a Mac or a Linux system, then you can easily achieve the same in Python without having to worry about changing the code. This is not possible in other programming languages, thus making Python one of the most portable languages available in the industry.

8. Highly Dynamic: As mentioned in an earlier paragraph, Python is one of the most dynamic languages available in the industry today. What this basically means is that the type of a variable is decided at the run time and not in advance. Due to the presence of this feature, we do not need to specify the type of the variable during coding, thus saving time and increasing efficiency.

Features of Python (Cont.)

9. Extensive Array of Library: Out of the box, Python comes inbuilt with a large number of libraries that can be imported at any instance and be used in a specific program. The presence of libraries also makes sure that you don’t need to write all the code yourself and can import the same from those that already exist in the libraries.

10. Support for Other Languages: Being coded in C, Python by default supports the execution of code written in other programming languages such as Java, C, and C#, thus making it one of the versatile in the industry.

Application of Python

1. Web Development

- Python can be used to make web-applications at a rapid rate. Why is that? It is because of the frameworks Python uses to create these applications. There is *common-backend logic* that goes into making these frameworks and a number of libraries that can help integrate protocols such as HTTPS, FTP, SSL etc. and even help in the processing of JSON, XML, E-Mail and so much more.
- Some of the most well-known frameworks are Django, Flask, Pyramid.
- Why use a framework?
- The ***security***, ***scalability***, ***convenience*** that they provide is unparalleled compared to starting the development of a website from scratch.

Application of Python (Cont.)

2. Game Development

- Python is also used in the development of interactive games. There are libraries such as PySoy which is a 3D game engine supporting Python 3, PyGame which provides functionality and a library for game development. Games such as Civilization-IV, Disney's Toontown Online, Vega Strike etc. have been built using Python.

Application of Python (Cont.)

3. Machine Learning and Artificial Intelligence

- Machine Learning and Artificial Intelligence are the talks of the town as they yield the most promising careers for the future. We make the computer learn based on past experiences through the data stored or better yet, create algorithms which makes the computer learn by itself. The programming language that mostly everyone chooses? It's Python. Why? Support for these domains with the **libraries** that exist already such as Pandas, Scikit-Learn, NumPy and so many more.
- Learn the algorithm, use the library and you have your solution to the problem. It is that simple. But if you want to go the hardcore way, you can design your own code which yields a better solution, which still is much easier compared to other languages.

Application of Python (Cont.)

4. Data Science and Data Visualization

- Data is *money* if you know how to extract relevant information which can help you take calculated risks and increase profits. You study the data you have, perform operations and extract the information required. Libraries such as Pandas, NumPy help you in extracting information.
- You can even visualize the data libraries such as Matplotlib, Seaborn, which are helpful in **plotting graphs** and much more. This is what Python offers you to become a Data Scientist.

Application of Python (Cont.)

5. Desktop GUI

- Python can be used to program **desktop applications**. It provides the Tkinter library that can be used to develop user interfaces. There are some other useful toolkits such as the wxWidgets, Kivy, PYQT that can be used to create applications on several platforms.
- You can start out with creating simple applications such as Calculators, To-Do apps and go ahead and create much more complicated applications.

Application of Python (Cont.)

6. Web Scraping Applications

- Python can be used to pull a large amount of data from websites which can then be helpful in various real-world processes such as price comparison, job listings, research and development and much more.
- Python has a library called BeautifulSoup which can be used to pull such data and be used accordingly.

Application of Python (Cont.)

7. Business Applications

- Business Applications are different than our normal applications covering domains such as e-commerce, ERP and many more. They require applications which are scalable, extensible and easily readable and Python provides us with all these features. Platforms such as Tryton can be used to develop such business applications.

Application of Python (Cont.)

8. Audio and Video Applications

- Python can be used to develop applications that can multi-task and also output media. Video and audio applications such as TimPlayer, Cplay have been developed using Python libraries and they provide better stability and performance compared to other media players.

Application of Python (Cont.)

9. CAD Applications

- Computer-Aided Designing is a very complicated application to make as many things have to be taken care of. Objects and their representation, functions are just the tip of the iceberg when it comes to something like this. Python makes this simple too and the most well-known application for CAD is Fandango.

Application of Python (Cont.)

10. Embedded Applications

- Python is based on C which means that it can be used to create Embedded C software for embedded applications. This helps us to perform higher-level applications on smaller devices which can compute Python.
- The most well-known embedded application could be the Raspberry Pi which uses Python for its computing. It can be used as a computer or like a simple embedded board to perform high-level computations.

Setting Path in Python

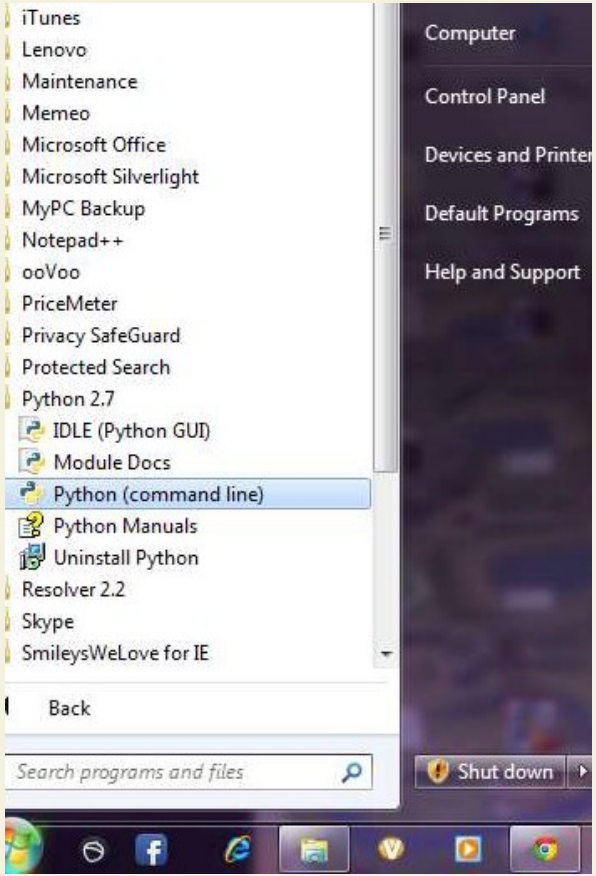
- Before starting working with Python, a specific path is to set.
- Your Python program and executable code can reside in any directory of your system, therefore Operating System provides a specific search path that index the directories Operating System should search for executable code.
- The Path is set in the Environment Variable of My Computer properties:
- To set path follow the steps: Right click on My Computer ->Properties ->Advanced System setting ->Environment Variable ->New
- In Variable name write path and in Variable value copy path up to C://Python(i.e., path where Python is installed). Click Ok ->Ok.

Installation of Python

- The easiest way to install Python is to download an installer for your computer from the official Python website, https://www.python.org.
- Python 3.9 is the latest version
- Python has official releases for Windows, Linux/Unix, and Mac OS X, as well as other platforms, including iOS (yes, you can write Python scripts on your favourite iPad or iPhone). If you are using Linux or Unix, Python is probably already available.


Python
PSF
Docs
PyPI
Jobs
Community
python
Search
GO
Socialize
About
Downloads
Documentation
Community
Success Stories
News
Events
Download the latest version for Windows
Download Python 3.6.3
Download Python 2.7.14
Wondering which version to use? Here's more about the difference between Python 2 and 3.
Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other
Want to help test development versions of Python? Pre-releases


Python 3.6.3 (64-bit) Setup
Install Python 3.6.3 (64-bit)
Select Install Now to install Python with default settings, or choose
Customize to enable or disable features.
Install Now
C:\Users\Jules\AppData\Local\Programs\Python\Python36
Includes IDLE, pip and documentation
Creates shortcuts and file associations
Customize installation
Choose location and features
python
for
windows
Install launcher for all users (recommended)
Add Python 3.6 to PATH
Cancel


iTunes
Lenovo
Maintenance
Memeo
Microsoft Office
Microsoft Silverlight
MyPC Backup
Notepad++
ooVoo
PriceMeter
Privacy SafeGuard
Protected Search
Python 2.7
IDLE (Python GUI)
Module Docs
Python (command line)
Python Manuals
Uninstall Python
Resolver 2.2
Skype
SmileysWeLove for IE
Back
Search programs and files
Computer
Control Panel
Devices and Printer
Default Programs
Help and Support
Shut down

First Python Program

- Basic syntax of Python to print hello world➔ print(“hello world”)
- Second Program to print User Information.

```
name = “Tanvi"
branch = “Artificial Intelligence"
age = “27"
print("My name is: ", name, )
print("My age is: ", age)
```

Indentation in Python

- Indentation is the most significant concept of the Python programming language. Improper use of indentation will end up **"IndentationError"** in our code.
- Indentation is nothing but adding whitespaces before the statement when it is needed. Without indentation Python doesn't know which statement to be executed to next. Indentation also defines which statements belong to which block. If there is no indentation or improper indentation, it will display "**IndentationError"** and interrupt our code.

Indentation in Python (Cont.)

```
def fun () :
        print (" Hello python")
        if condition :
                statement
        else:
                statement
statement
```

How the Python Intepreter Visualize

```
Code block 1 begins
        Code block 1 continues
        Code block 2 begins
                Code block 3 begins
        Code block 2 continues
                Code block 3 continues

Code block 1 continues
```

Comments in Python

- Comments are essential for defining the code and help us and other to understand the code. By looking the comment, we can easily understand the intention of every line that we have written in code. We can also find the error very easily, fix them, and use in other applications.
- Types of Comment
- Python provides the facility to write comments in two ways- single line comment and multi-line comment.
- **Single-Line Comment -** Single-Line comment starts with the hash # character followed by text for further explanation.

```
# defining the marks of a student
Marks = 90
```

Comments in Python (Cont.)

- We can also write a comment next to a code statement. Consider the following example.

  Name = “Tanvi" # the name of a professor is Tanvi

  Branch = "Computer Science" # defining branch

- **Multi-Line Comments -** Python doesn't have explicit support for multi-line comments but we can use hash # character to the multiple lines. For e.g.,

  # we are defining for loop

  # To iterate the given list.

  We can also use another way.

  " " "

  This **is** an example

  Of multi-line comment

  Using triple-quotes " " "

Variables in Python

- A variable is a memory location where a programmer can store a value. Example : roll_no, amount, name etc. Value is either string, numeric etc. Example : "Sara", 120, 25.36
- A variable is created the moment you first assign a value to it.
- The value stored in a variable can be accessed or updated later.

```
x = 5
y = “Tanvi"
print(x)
print(y)
```

- Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

```
x = 4          # x is of type int
x = “Tanvi" # x is now of type str
print(x)
```

Variables in Python (Cont.)

Casting

- If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3)      # x will be '3'
y = int(3)      # y will be 3
z = float(3)  # z will be 3.0
```

Get the Type

- You can get the data type of a variable with the type() function.

```
x = 5
y = “Tanvi"
print(type(x))
print(type(y))
```

Variables in Python (Cont.)

- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
- String variables can be declared either by using single or double quotes.
- Variable names are case-sensitive i.e., A and a are different.

Rules for Python variables:

- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables).
- There are some reserved words which you cannot use as a variable name because Python uses them for other things.

Variables in Python (Cont.)

Good Variable Name

- Choose meaningful name instead of short name. roll_no is better than rn.
- Maintain the length of a variable name. Roll_no_of_a-student is too long?
- Be consistent; roll_no or RollNo

Identifiers in Python

- Python identifiers are user-defined names. They are used to specify the names of variables, functions, class, module, etc.

Rules to Create Python 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 _. Names like myClass, var_1 and print_this_to_screen, all are valid example.
- An identifier cannot start with a digit. 1variable is invalid, but variable1 is a valid name.
- Python identifier can’t contain only digits.
- There is no limit on the length of the identifier name.
- Keywords cannot be used as identifiers e.g., import, if, else, break, continue, etc.

Identifiers in Python (Cont.)

Python Valid Identifiers Example

- **ab10c**: contains only letters and numbers
- **abc_DE**: contains all the valid characters
- **_:** surprisingly but Yes, underscore is a valid identifier
- **_abc**: identifier can start with an underscore

Python Invalid Identifiers Example

- **99**: identifier can’t be only digits
- **9abc**: identifier can’t start with number
- **x+y**: the only special character allowed is an underscore
- **for**: it’s a reserved keyword

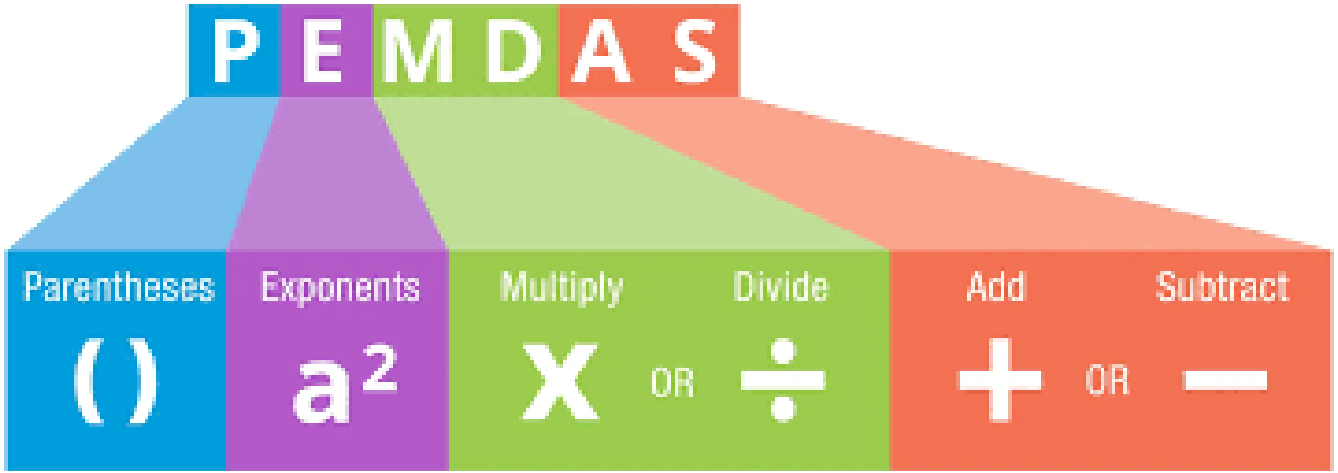
Operators in Python

- Arithmetic Operators

OPERATOR	DESCRIPTION	SYNTAX
+	Addition: adds two operands	x + y
-	Subtraction: subtracts two operands	x - y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when first operand is divided by the second	x % y
**	Power : Returns first raised to power second	x ** y

Operators in Python (Cont.)

- Order of Operation: The order in which a mathematical expression will be solved



Operators in Python (Cont.)

- Order of Operation: The order in which a mathematical expression will be solved

```
print (5 – 6 * 2)
print ((5-6) * 2)
```

```
print (3 ** 3 * 5)
print (3** (3 * 5))
```

Operators in Python (Cont.)

- **Relational / Comparison Operator**

OPERATOR	DESCRIPTION	SYNTAX
>	Greater than: True if left operand is greater than the right	x > y
<	Less than: 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 right	x <= y

Operators in Python (Cont.)

- **Assignment Operator**

Operator	Example	Equivalent to
=	x = 5	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 **= 5	x = x ** 5

Operators in Python (Cont.)

- **Logical Operator**

OPERATOR	DESCRIPTION	SYNTAX
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if operand is false	not x

Operators in Python (Cont.)

- **Bitwise Operator**

OPERATOR	DESCRIPTION	SYNTAX
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>
<<	Bitwise left shift	x<<

Operators in Python (Cont.)

- **Bitwise AND (&)**

a, b = 4, 5

print (a & b) ➔ Output: 4

8 4 2 1

==========

a = 0 1 0 0 (binary representation of the value 4 in 4 bits)

b = 0 1 0 1 (binary representation of the value 5 in 4 bits)

0100 (binary representation of the value 4 in 4 bits)

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

Operators in Python (Cont.)

- **Bitwise OR (|)**

a, b = 4, 5

print (a & b) ➔ Output: 5

8 4 2 1

==========

a = 0 1 0 0 (binary representation of the value 4 in 4 bits)

b = 0 1 0 1 (binary representation of the value 5 in 4 bits)

0101 (binary representation of the value 5 in 4 bits)

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

Operators in Python (Cont.)

- **Bitwise XOR (^)**

a, b = 4, 5

print (a ^ b) ➔ Output: 1

a = 0100 (binary representation of the value 4 in 4 bits)

b = 0101 (binary representation of the value 5 in 4 bits)

0001 (binary representation of the value 1 in 4 bits)

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Operators in Python (Cont.)

BitwiseNOT (~): Returns one's complement of the number.

a = 10 = 1010 (Binary)

~a = ~1010

= -(1010 + 1)

= -(1011)

= -11 (Decimal)

Operators in Python (Cont.)

Bitwise Right shift (>>)

a = 7

print (a >> 1) ➔ Output: 3

0 1 1 1

---->---->---->---->

0 0 1 1

Operators in Python (Cont.)

Bitwise Left shift (<<)

a = 7

print (a << 1) ➔ Output: 14

0 1 1 1

<-------<-------<-------<-------

1 1 1 0

Special Operators

Membership Operator

- Python membership operators are used to check the membership of value inside a Python data structure. If the value is present in the data structure, then the resulting value is true otherwise it returns false.

Operator	Meaning	Example
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

Special Operators (Cont.)

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.

Operator	Meaning	Example
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

Python print() function

- The print() function prints the specified message to the screen, or other standard output device.
- The message can be a string, or any other object, the object will be converted into a string before written to the screen.

Syntax: print(*object(s)*, sep=*separator*, end=*end*, file=*file*, flush=*flush*)

Parameter	Description
object(s)	Any object, and as many as you like. Will be converted to string before printed
sep='separator'	Optional. Specify how to separate the objects, if there is more than one. Default is ' '
end='end'	Optional. Specify what to print at the end. Default is '\n' (line feed)
File	Optional. An object with a write method. Default is sys.stdout
Flush	Optional. A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False

Python print() function (Cont.)

Print a message onto the screen:

- print("Hello World")

Print two messages, and specify the separator:

- print("Hello", "how are you?", sep="---")

Python type() function

- The type() function either returns the type of the object or returns a new type object based on the arguments passed.

Syntax**:** type(*object, bases, dict*)

Parameter	Description
Object	Required. If only one parameter is specified, the type() function returns the type of this object
Bases	Optional. Specifies the base classes
dict	Optional. Specifies the namespace with the definition for the class

Python type() function (Cont.)

Example: Return the type of these objects:

- a = ('apple', 'banana', 'cherry')
  b = "Hello World"
  c = 33

  x = type(a) ➔ <class 'tuple'>
  y = type(b) ➔ <class 'str'>
  z = type(c) ➔ <class 'int'>

Python id() function

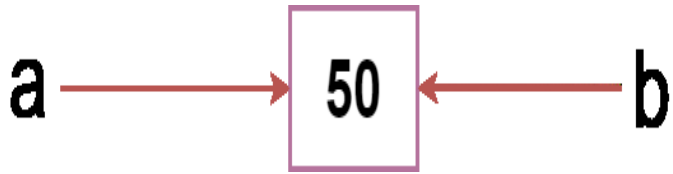
- The id() function returns a unique id for the specified object.
- All objects in Python has its own unique id.
- The id is assigned to the object when it is created.
- The id is the object's memory address, and will be different for each time you run the program. (except for some object that has a constant unique id, like integers from -5 to 256)
- Syntax**:** id(*object*)

Parameter	Description
Object	Any object, String, Number, List, Class etc.

Python id() function (Cont.)

Example: Return the unique id of a tuple object:

- x = 100
  y = id(x)
  a = 50
  b = a
  print(id(a)) ➔ 140734982691168
  print(id(b)) ➔ 140734982691168
  # Reassigned variable a
  a = 500
  print(id(a)) ➔ 2822056960944



User input in Python

- Developers often have a need to interact with users, either to get data or to provide some sort of result. Most programs today use a dialog box as a way of asking the user to provide some type of input. While Python provides us with two inbuilt functions to read the input from the keyboard: raw_input() and input(). The results can be stored into a variable.
- **raw_input()** – It reads the input or command and returns a string.
- **input()** – Reads the input and returns a python type like list, tuple, int, etc.

User input in Python (Cont.)

- When you use input() function, Python automatically converts the data type based on your input. So input() function can return anything int, str, float, bool, list etc based on your input. For instance,

  val = input("Enter any value: ")

  Enter any value: 7

  type(val) ➔ int

  val = input("Enter any value: ")

  Enter any value: 7.0

  type(val) ➔ float

  val = input("Enter any value: ")

  Enter any value: 'abc'

  type(val) ➔ str

  val = input("Enter any value: ")

  Enter any value: True

  type(val) ➔ bool

User input in Python (Cont.)

- When you use raw_input(), Python will simply return you a string irrespective of your input value data type.

  val = raw_input("Enter any value: ")

  Enter any value: 7

  type(val) ➔ str

  val = raw_input("Enter any value: ")

  Enter any value: 7.0

  type(val) ➔ str

  val = raw_input("Enter any value: ")

  Enter any value: 'abc'

  type(val) ➔ str

User input in Python (Cont.)

Python 2:

- raw_input() takes exactly what the user typed and passes it back as a string.
- input() first takes the raw_input() and then performs an eval() on it as well. It means that python automatically identifies whether you entered a number, string or even a list.

Python 3:

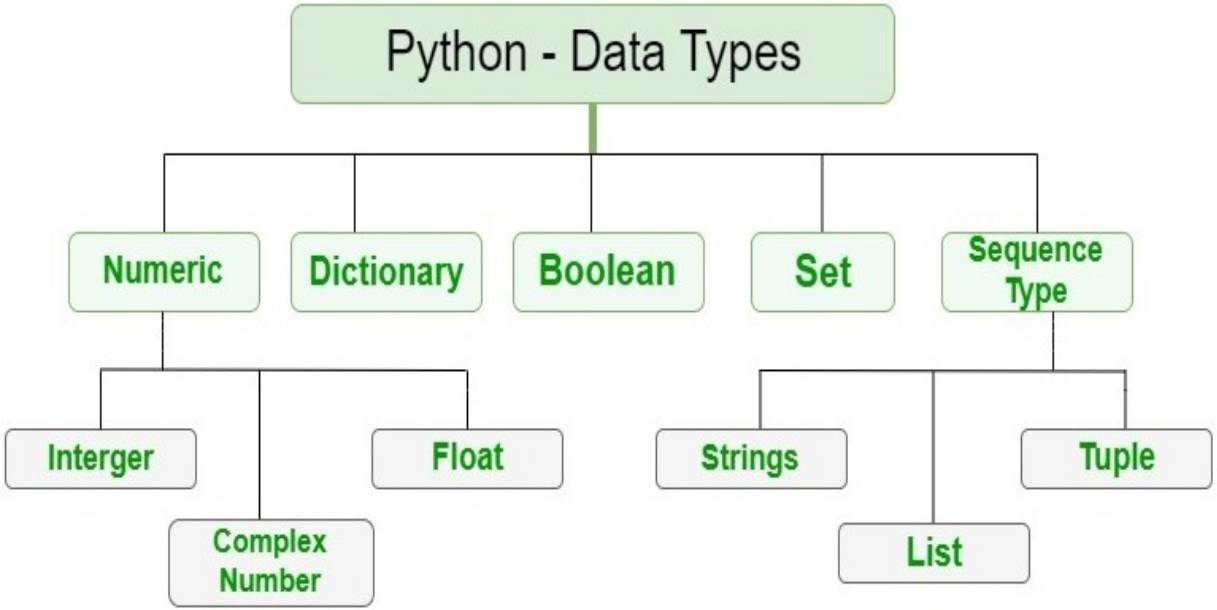
- raw_input() was renamed to input() so now input() returns the exact string.
- Old input() was removed.
- If you want to use the old input(), meaning you need to evaluate a user input as a python statement, you have to do it manually by using eval(input()).

Data Types

- Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.
- Following are the standard or built-in data type of Python:

1. **Numeric**
2. **Sequence Type**
3. **Boolean**
4. **Set**
5. **Dictionary**

Data Types (Cont.)



Data Types (Cont.)

In Python, numeric data type represent the data which has numeric value. Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.

1. Integers – This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.

2. Float – This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.

3. Complex Numbers – Complex number is represented by complex class.

It is specified as *(real part)* + *(imaginary part)j*. For example – 2+3j

- **Note** – type() function is used to determine the type of data type.

Data Types (Cont.)

Sequence Type

- In Python, sequence is the ordered collection of similar or different data types. Sequences allows to store multiple values in an organized and efficient fashion. There are several sequence types in Python –

1. String
2. List
3. Tuple

Data Types (Cont.)

Boolean

- Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). But non-Boolean objects can be evaluated in Boolean context as well and determined to be true or false. It is denoted by the class bool.
- **Note** – True and False with capital ‘T’ and ‘F’ are valid booleans otherwise python will throw an error.

Mutable and Immutable

- A **mutable object** is a changeable object that can be modified after it is created.
- An **immutable object** is an object whose state can never be modified after it is created.

List of Mutable and Immutable objects

- **Mutable objects:** list, dict, set, byte array
- **Immutable objects:** int, float, complex, string, tuple, frozen set, bytes

Python Strings

- A string is a sequence of characters.
- A character is simply a symbol. For example, the English language has 26 characters.
- Computers do not deal with characters, they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0s and 1s.

Create a string in Python

- Strings can be created by enclosing characters inside a single quote or double-quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

  # all of the following are equivalent

  my_string = 'Hello' ➔ Hello

  my_string = "Hello" ➔ Hello

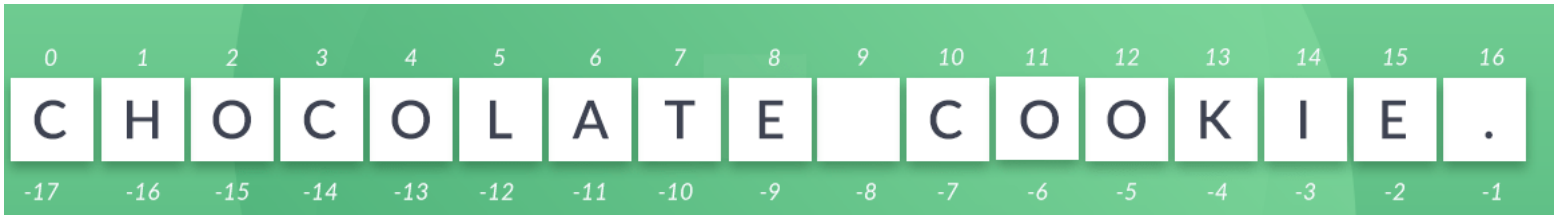
  my_string = '"Hello"' ➔ Hello

  # triple quotes string can extend multiple lines

  my_string = """Hello, welcome to the world of Python""" ➔ Hello, welcome to
  the world of Python

Access characters in a string (Cont.)

- We can access individual characters using indexing and a range of characters using slicing.
- Index starts from 0. Trying to access a character out of index range will raise an IndexError.
- The index must be an integer. We can't use floats or other types, this will result into TypeError.
- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on.



Access characters in a string (Cont.)

- We can access a range of items in a string by using the slicing operator :(colon).

  Example:

  #Accessing string characters in Python

  str = 'python_program'

  print('str = ', str) ➔ str = python_program

  #first character

  print('str[0] = ', str[0]) ➔ str[0] = p

  #last character

  print('str[-1] = ', str[-1]) ➔ str[-1] = m

Access characters in a string (Cont.)

Slicing Strings

- You can return a range of characters by using the slice syntax.
- Specify the start index and the end index, separated by a colon, to return a part of the string.
- Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!“

#slicing 2nd to 5th character
print(b[1:5]) ➔ ello

#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2]) ➔ str[5:-2] = , Worl
```

Access characters in a string (Cont.)

Slice From the Start

- By leaving out the start index, the range will start at the first character:

Example: Get the characters from the start to position 5 (not included):

```
b = "Hello, World!"
print(b[:5]) ➔ Hello
```

Slice To the End

- By leaving out the *end* index, the range will go to the end:

Example: Get the characters from position 2, and all the way to the end:

```
b = "Hello, World!"
print(b[2:]) ➔ llo, World!
```

Access characters in a string (Cont.)

- String slicing can also accept a third parameter, the **stride**, which refers to how many characters you want to move forward after the first character is retrieved from the string. The value of stride is set to 1 by default.
- Let's see stride in action to understand it better:

  number_string = "1020304050"

  print(number_string[0:-1:2]) ➔ 12345

  print(number_string[::-1]) ➔ 0504030201
- The value of -1 for the stride allows you to start from the end character and then move one character at a time.
- Alternatively, if you provide -2 as a value, you start from the end character and move two characters at a time.

Change or delete a string

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

  my_string = ‘pythonprogram'

  my_string[5] = ‘a’ ➔ TypeError: 'str' object does not support item assignment

  my_string = ‘Python’

  my_string ➔ ‘Python’

- We cannot delete or remove characters from a string. But deleting the string entirely is possible using the del keyword.

  del my_string[1] ➔ TypeError: 'str' object doesn't support item deletion

  del my_string

  print(my_string) ➔ NameError: name ‘my_string' is not defined

Modify Strings

- Python has a set of built-in methods that you can use on strings.

Upper Case: The upper() method returns the string in upper case:

Example

- a = "Hello, World!"
  print(a.upper()) ➔ HELLO, WORLD

Lower Case: The lower() method returns the string in lower case:

Example

- a = "Hello, World!"
  print(a.lower()) ➔ hello, world!

Modify Strings (Cont.)

Remove Whitespace

- Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

Example

- The strip() method removes any whitespace from the beginning or the end:
- a = " Hello, World! "
  print(a.strip()) # returns "Hello, World!"

Modify Strings (Cont.)

Replace String: The replace() method replaces a string with another string:

Example

- a = "Hello, World!"
  print(a.replace("H", "J")) ➔ Jello, World!

Split String

- The split() method returns a list where the text between the specified separator becomes the list items.

Example

- The split() method splits the string into substrings if it finds instances of the separator:
- a = "Hello|World!"
  print(a.split("|")) ➔ ['Hello', ' World!']

Concatenation of Two or More Strings

- There are many operations that can be performed with strings which makes it one of the most used data types in Python.

- Joining of two or more strings into a single one is called concatenation.
- The **+** operator does this in Python. Simply writing two string literals together also concatenates them.
- The * operator can be used to repeat the string for a given number of times.

Concatenation of Two or More Strings (Cont.)

```
str1 = 'Hello'
str2 ='World!'
# using +
print('str1 + str2 = ', str1 + str2) ➔ HelloWorld!
# using *
print('str1 * 3 =', str1 * 3) ➔HelloHelloHello
# two string literals together
print('Hello "World!‘)  ➔ Hello “World!
```

Concatenation of Two or More Strings (Cont.)

Example

- To add a space between them, add a " ":
- a = "Hello"
  b = "World"
  c = a + " " + b
  print(c) ➔ Hello World

String Length

- To get the length of a string, use the len() function.

Example:

- The len() function returns the length of a string:

```
a = "Hello, World!"
print(len(a))
```

String Format

- To make sure a string will display as expected, we can format the result with the format() method.
- Will this run??

  age = 27
  txt = "My name is Tanvi, I am " + age
  print(txt) ➔ TypeError: must be str, not int
- We can combine strings and numbers by using the format() method!
- The format() method takes the passed arguments, formats them, and places them in the string.

String Format (Cont.)

Example: Use the format() method to insert numbers into strings:

- age = 27
  txt = "My name is Tanvi, and I am {}"
  print(txt.format(age))
- The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

Example

- quantity = 3
  itemno = 567
  price = 49.95
  myorder = "I want {} pieces of item {} for {} ruppes."
  print(myorder.format(quantity, itemno, price))

String Format (Cont.)

- You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

Example

- quantity = 3
  itemno = 567
  price = 49.95
  myorder = "I want to pay {2} ruppes for {0} pieces of item {1}."
  print(myorder.format(quantity, itemno, price))

String Format (Cont.)

Named Indexes

- You can also use named indexes by entering a name inside the curly brackets {carname}, but then you must use names when you pass the parameter values txt.format(carname = "Ford"):

Example

- myorder = "I have a {carname}, it is a {model}."
  print(myorder.format(carname = "Ford", model = "Mustang"))

Example

- Insert the price inside the placeholder, the price should be in fixed point, two-decimal format:
- txt = "For only {price:.2f} dollars!"
  print(txt.format(price = 49))

Escape Characters

- To insert characters that are illegal in a string, use an escape character.
- An escape character is a backslash(\) followed by the character you want to insert.
- An example of an illegal character is a double quote inside a string that is surrounded by double quotes:
- txt = ”We are the so-called "Vikings" from the north.”
- To fix this problem, use the escape character (\“):

Example

- The escape character allows you to use double quotes when you normally would not be allowed:
- txt = ”We are the so-called \"Vikings\" from the north.”
- print(txt) ➔ We are the so-called "Vikings" from the north.

String Methods

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
startswith()	Returns true if the string starts with the specified value
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string

String Methods (Cont.)

Method	Description
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa

String Methods (Cont.)

Method	Description
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
lower()	Converts a string into lower case
maketrans()	Returns a translation table to be used in translations
replace()	Returns a string where a specified value is replaced with a specified value
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list

Thank You