

COURSE:- Python Programming

Sem:- I

Module: 1, 2

By:

Dr. Ritu Jain,

Associate Professor,

Computer Science-ISU Engineering

Introduction

- Python is a general purpose programming language conceived in 1989 by Dutch programmer Guido van Rossum while working at National Research Institute at Netherlands.
- But officially Python was made available to public in 1991. The official Date of Birth for Python is : Feb 20th 1991.
- Python was named after the BBC TV Show *Monty Python's Flying Circus*.
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell.
- Most of syntax in Python Derived from C and ABC language.



Applications of Python

- The application areas of python are numerous. Python is used:
 1. For developing Desktop Applications
 2. For developing Web Applications
 3. For developing Database Applications
 4. For Network Programming
 5. For developing games
 6. For Machine Learning and Data Analysis Applications
 7. For developing Artificial Intelligence Applications
 8. For IOT and many more

Top Companies using Python

- Industrial Light and Magic
- Google
- Facebook
- Instagram
- Spotify
- Quora
- Netflix
- Dropbox
- Reddit and many more

Created by Dr. Ritu Jain

Python

- Python is free and open source language, with development coordinated through the Python Software Foundation.
- Python is one of the most popular programming languages.
- Python is an all-purpose programming language that can be used to create desktop applications, 3D graphics, video games, and even websites.
- It is easy to learn and it's simpler than languages like C, C++, or Java.
- Python is powerful and robust enough to create advanced applications.
- This makes Python a good language for students with or without any programming experience.

Development Steps of Python

- Guido Van Rossum published the first version of Python code (version 0.9.0) in February 1991. This release included exception handling, functions, and the core data types such as list, dictionary, string and others. It was also object oriented and had a module system.
- Python version 1.0 was released in January 1994. The major new features included in this release were the functional programming tools such as lambda, map, filter and reduce.
- Six and a half years later in October 2000, Python 2.0 was introduced. This release included list comprehensions, a full garbage collector and it was supporting Unicode.
- Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 (also known as "Python 3000" and "Py3K") was released.
- **Python 3 is not backwards compatible with Python 2.x.** The emphasis in Python 3 had been on the removal of duplicate programming constructs and modules.
- Python 3 version - Python 3.12.6

Official Introduction to Python

- Python is an easy to learn, powerful programming language.
- It has efficient high-level data structures and a simple but effective approach to object-oriented programming.
- 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.

Features of Python

- **Easy to code:**
 - Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, Java, etc.
 - Supports quick development
 - Short Code => Less to type, debug, maintain
 - Readability => Maintainable
- **Free and Open Source:**
 - Freeware: Python language is freely available at the official website www.python.org. You don't have to pay anything for using it.
 - It is open-source, this means that source code is also available to the public. So you can download it, change it, use it, and distribute it.
- **Expressive**
 - Python needs to use only a few lines of code to perform complex tasks. For example, to display Hello World, you simply need to type one line - `print("Hello World")`. Other languages like Java or C would take up multiple lines to execute this.

Features of Python

- **Object-Oriented and Procedure-Oriented**
 - A programming language is object-oriented if it focuses design around data and objects, rather than functions and logic. On the contrary, a programming language is procedure-oriented if it focuses more on functions (code that can be reused). One of the critical features of Python is that it supports both object-oriented and procedure-oriented programming.
- **GUI Programming Support:**
 - One of the key aspects of any programming language is support for GUI or Graphical User Interface. A user can easily interact with the software using a GUI. Python offers various toolkits, which allows for GUI's easy and fast development.

Features of Python

- **High-Level Language:**

- Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

- **Extensible feature:**

- Programs in other languages can be used in python. If performance is very important, so can write code in C, C++ and extend it to python. This makes Python an extensible language, meaning that it can be extended to other languages.

- **Python is Portable language:**

- Python language is also a portable language. For example, if we have python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

- **Embeddable**

- We just saw that we can put code in other languages in our Python source code. However, it is also possible to put our Python code in a source code in a different language like C++. This allows us to integrate scripting capabilities into our program of the other language.

Features of Python

- **Interpreted Language:**

- Python is an Interpreted Language because Python code is executed line by line at a time. There is no need to compile python code this makes it easier to debug our code. The source code of python is converted into an intermediate form called bytecode.

- **Large Standard Library**

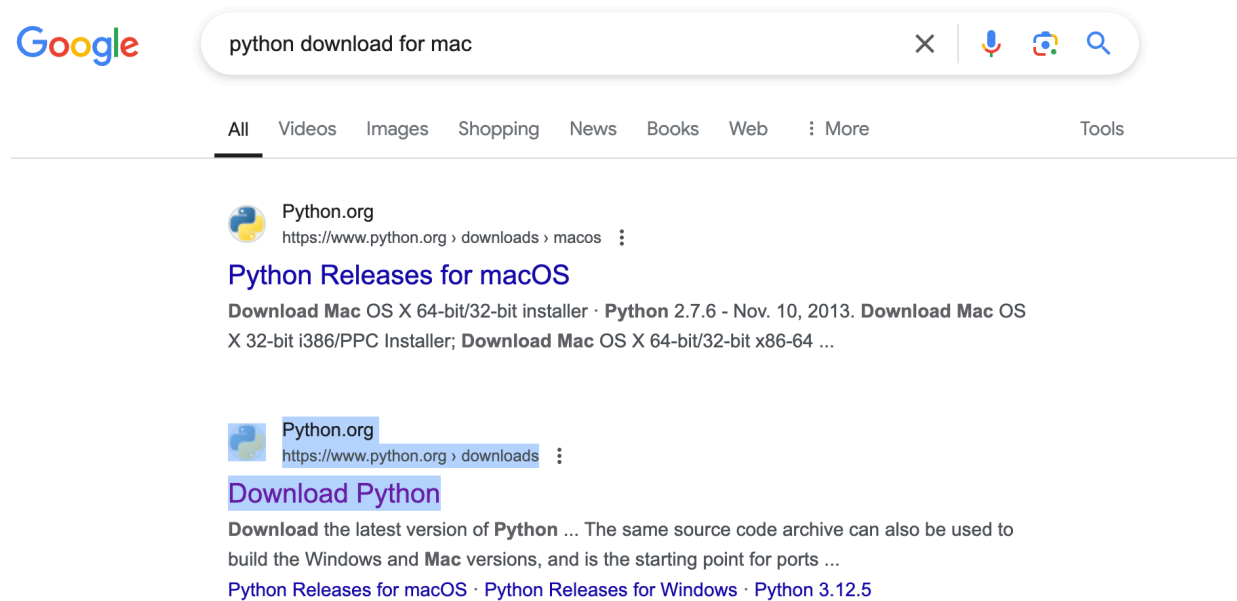
- Python has a large standard library which provides a rich set of module and functions so you do not have to write your own code for every single thing.

- **Dynamically Typed Language:**

- Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don't need to specify the type of variable.

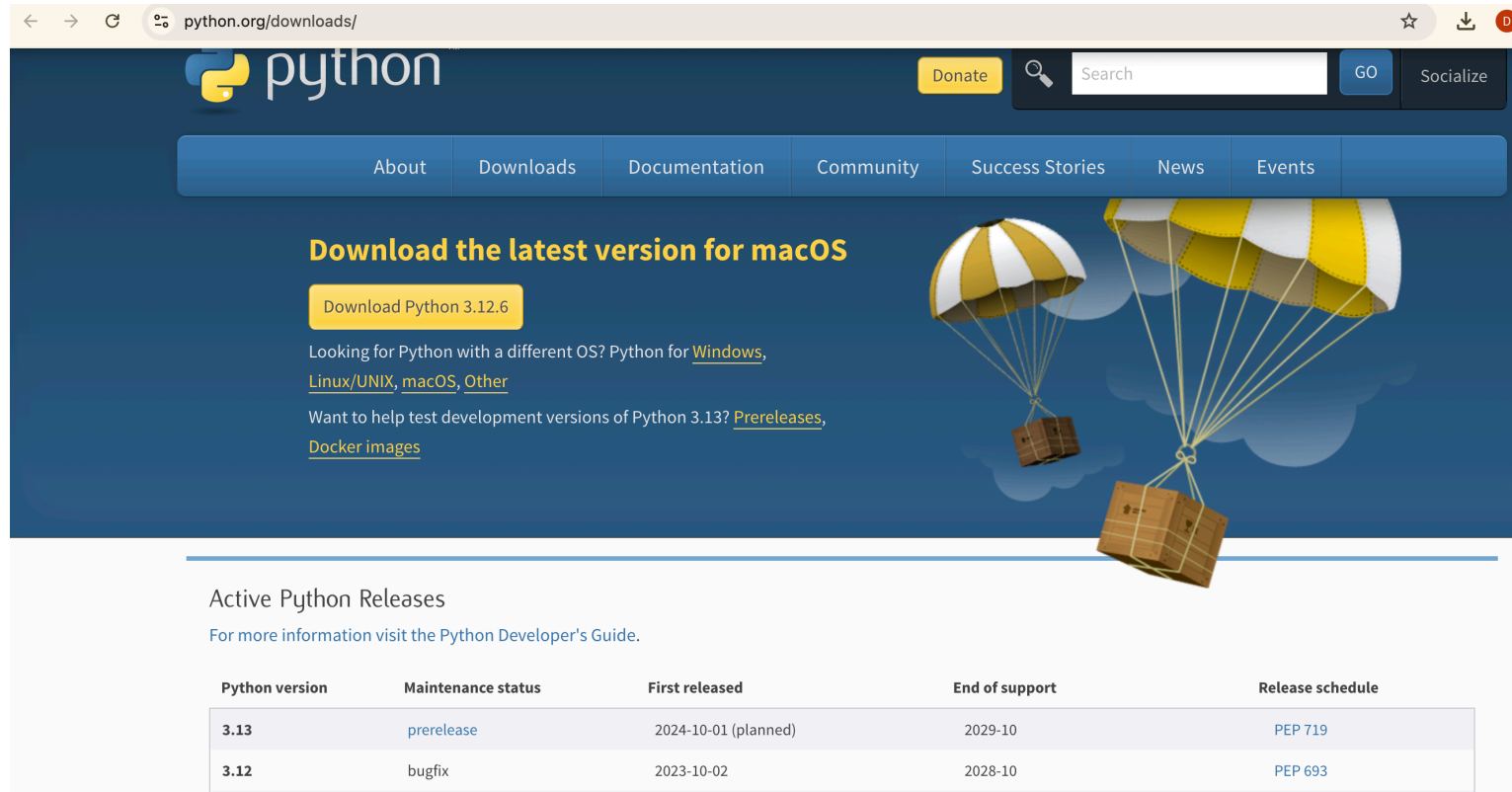
Installation of Python

- Version: Python 3.12.6



Created by Dr. Ritu Jain

Installation of Python



The screenshot shows the Python.org/downloads page. The header includes the Python logo, a 'Donate' button, a search bar, and a 'GO' button. The navigation menu has links for About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area features a large banner with the text 'Download the latest version for macOS' and a yellow button 'Download Python 3.12.6'. Below this, there are links for 'Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other' and 'Want to help test development versions of Python 3.13? Prereleases, Docker images'. The bottom section is titled 'Active Python Releases' and contains a table with the following data:

Python version	Maintenance status	First released	End of support	Release schedule
3.13	prerelease	2024-10-01 (planned)	2029-10	PEP 719
3.12	bugfix	2023-10-02	2028-10	PEP 693

- Choose the following:

Created by Dr. Ritu Jain

3.12	bugfix	2023-10-02	2028-10	PEP 693
------	--------	------------	---------	---------

- So now, you can reach Python in the following ways:
- **Terminal:** You can run Python on terminal
 - Search for terminal, and type the following: python
 - Now, you can use it as an interpreter. As an example, we have calculated 2+3.

```
(base) dr.ritujain@DrRitus-MacBook-Air ~ % python
Python 3.12.4 | packaged by Anaconda, Inc. | (main, Jun 18 2024, 10:07:17) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> 3
```

```
3
```

```
>>> 3+2
```

```
5
```

```
>>> █
```

Created by Dr. Ritu Jain

- **Writing Your First Python Program**

- Create a folder “Python_Prg”. You will be storing all your Python programs in this folder.
- In Notepad type in your python program
- *Save your notepad program with extension .py (eg. try2_notepad.py). In the field Save as type select All Files. Click on Save.*

- **Running Your First Program**

- Open command prompt. Go to the folder where you have saved the file.
- To run the program, type *python Hello.py* and hit Enter.
- You should see the line *Hello World!*

```
C:\Users\Ritu>cd\
```

Created by Dr. Ritu Jain

```
C:\>cd Rit_py_prg
```

```
C:\Rit_py_prg>python try2_notepad.py  
Created this file through notepad
```

Python IDLE

- **The IDLE :** There are various editors for writing Python code.
- The easiest way to start writing Python is to use IDLE, an integrated development environment for Python.
- IDLE comes as part of the standard Python installation package. IDLE is an application, just like any other application on your computer. Start it the same way you would start any other application, e.g., by double-clicking on an icon.
- Using IDLE you get some help with the syntax; as an example, keywords are displayed in a different color.
- Open the Applications. Click on IDLE . This will take you to the command prompt for Python.
- To run a script with multiple lines, write the program in the editor in Python IDLE and save it in the directory that you would use to save all programs. It will be saved as a *.py for example hello.py file. You can run the script from terminal by typing python hello.py.

Python IDLE

```
(base) dr.ritujain@DrRitus-MacBook-Air ~ % python
Python 3.12.4 | packaged by Anaconda, Inc. | (main, Jun 18 2024, 10:07:17) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 3
3
>>> 3+2
5
>>> █
```

Created by Dr. Ritu Jain

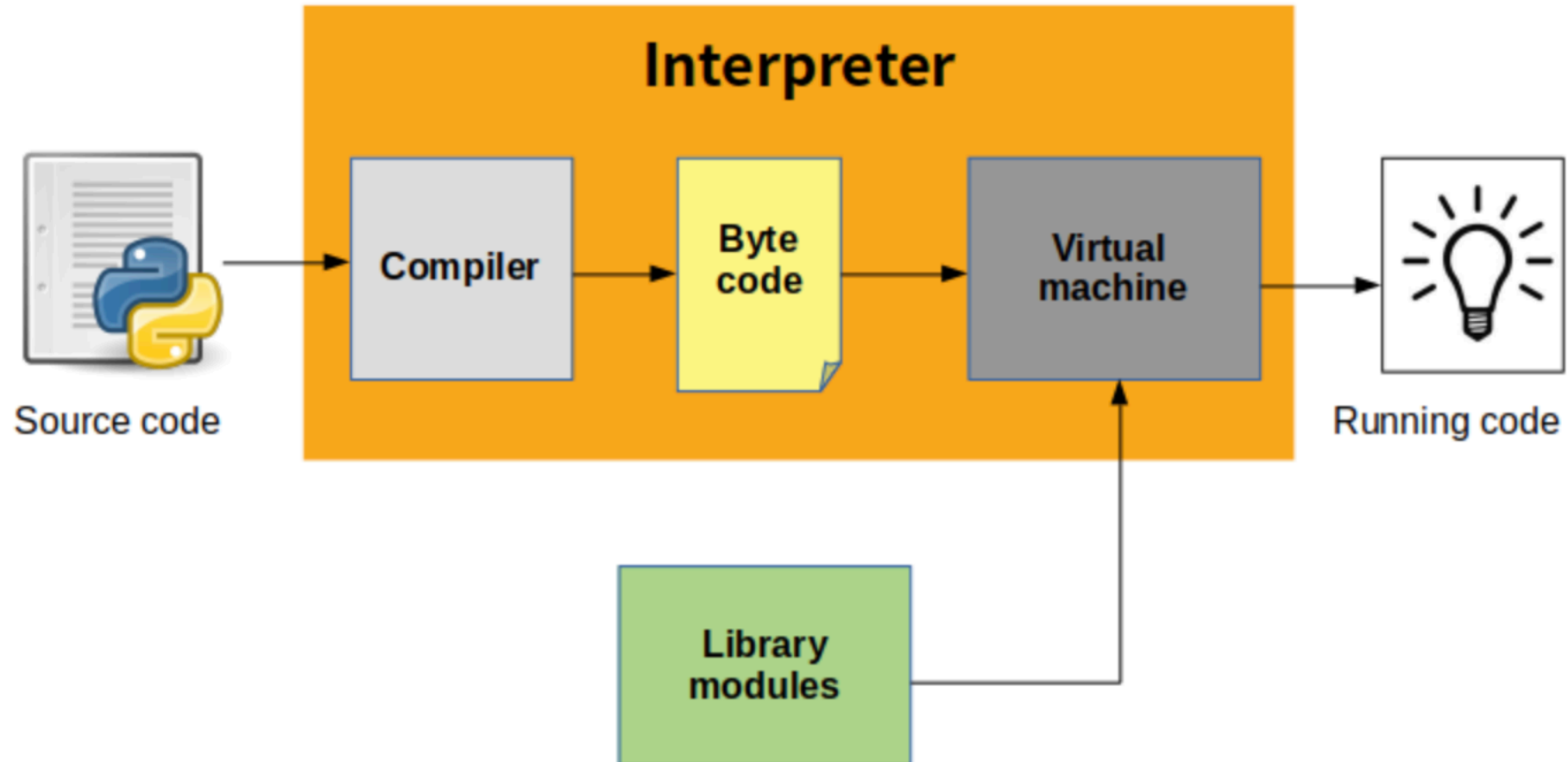
- Click New File.
- Type:
 - `a=2`
 - `b=3`
 - `print(a+b)`
- Save it as `add.py`.
- Click run module:

```
nano ~/ .bash_profile
```

```
alias python=python3
```

Created by Dr. Ritu Jain

Python is compiled as well as interpreted



Keywords

- Keywords are the reserved words in Python.
- All the keywords except True, False, and None are in lower case.
- In your terminal, to check the keywords:

```
(base) dr.ritujain@DrRitus-MacBook-Air ~ % python  
Python 3.12.4 | packaged by Anaconda, Inc. | (main, Jun 18 2024, 10:07:17) [Clang 14.0.6 ] on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import keyword  
>>> keyword.kwlist  
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Identifiers

- It is a name given to entities like variables, class, functions, etc.
- **Rules for writing identifiers**
 - **a to z, A to Z , 0 to 9, underscore (_)** are allowed.
 - Variable names can be arbitrarily long.
 - Keywords cannot be used as identifiers.
 - Cannot use special symbols except _.
 - Spaces are not allowed.
 - **Identifier should not starts with digit**
 - Case-sensitive language. Therefore, **count** and **Count** are different.
 - It is legal to use uppercase letters, but it is a good idea to begin variable names with a lowercase letter.
- **Good Programming practice**
 - Variable names should be short but descriptive.

Valid and Invalid Variable Names

Table 1-3: Valid and Invalid Variable Names

Valid variable names	Invalid variable names
balance	current-balance (hyphens are not allowed)
currentBalance	current balance (spaces are not allowed)
current_balance	4account (can't begin with a number)
_spam	42 (can't begin with a number)
SPAM	total_\$um (special characters like \$ are not allowed)
account4	'hello' (special characters like ' are not allowed)

Variable

- **A variable comes into existence as a result of assigning a value to it.** Unlike in other languages, you don't need to declare it in any special way.
- If you assign any value to a nonexistent variable, the variable will be **automatically created**.
- Eg: `a=10`
`print(a)`

```
a=10  
balance=10000.5  
• client_name='John'
```


Variables

- In Python, variables are seen as labels or tags that are tied to some value. Python considers the values as objects. Thus, **Variables provide a way to associate names with objects.**
- For example, the statement `a=1` means the value '1' is created first in memory and then a tag by the name 'a' is created.

• Eg:

- `pi = 3`
- `radius = 11`
- `area = pi * (radius**2)`
- `radius = 14`

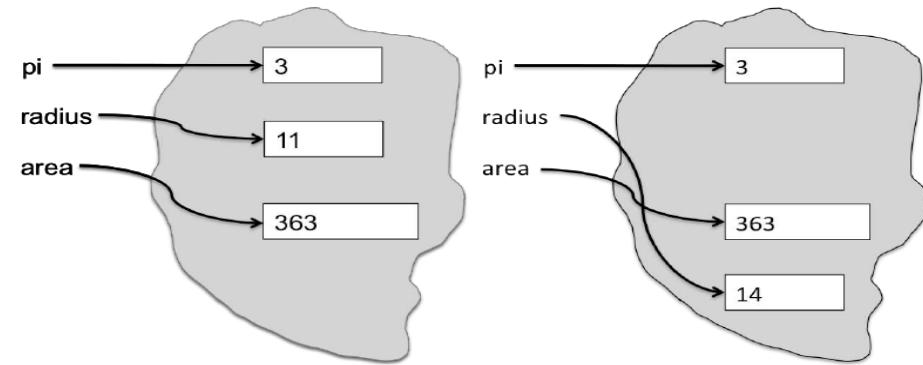


Figure Binding of variables to objects

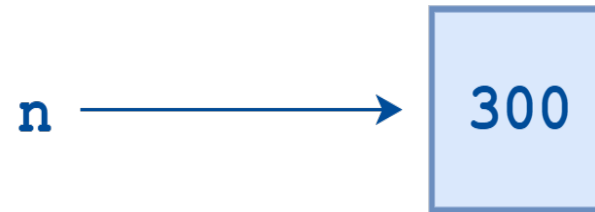
- It first binds the names `pi` and `radius` to different objects of type `int`. It then binds the name `area` to a third object of type `int`. This is depicted in the left panel of Figure.
- If the program then executes `radius = 14`, the name `radius` is rebound to a different object of type `int`, as shown in the right panel of Figure.
- Note that this assignment has no effect on the value to which `area` is bound. It is still bound to the object denoted by the expression `3*(11**2)`.

Variable

- The assignment statement binds a *name*, on the left-hand side of the operator, to a *value*, on the right-hand side.
- In Python, **a variable is just a name**, nothing more.
- An **assignment** statement associates the name to the left of the = symbol with the object denoted by the expression to the right of the =.
- **An object can have one, more than one, or no name associated with it.**
- Dynamic typing:
 - All type checking is done at runtime.
 - No need to declare a variable or give it a type before use.

Multiple References to a Single Object

```
>>> n = 300
>>> print(n)
300
>>> type(n) # int object
<class 'int'>
```



- What happens when `m=n` is executed? Python does not create another object. It simply creates a new symbolic name or reference, `m`, which points to the same object that `n` points to.

```
>>> m = n
```

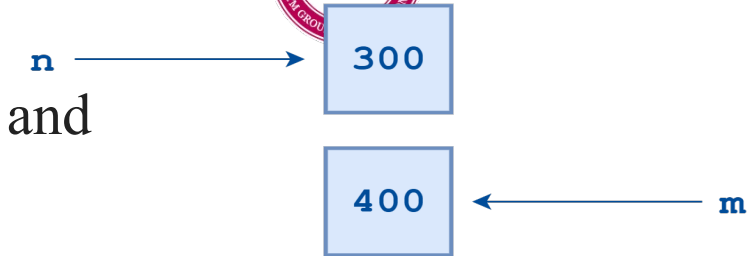


There is one memory reference by two names. **Python is using memory efficiently**

- Next, suppose you do this:

- `>>> m = 400`

- Now Python creates a new integer object with the value 400, and m becomes a reference to it.

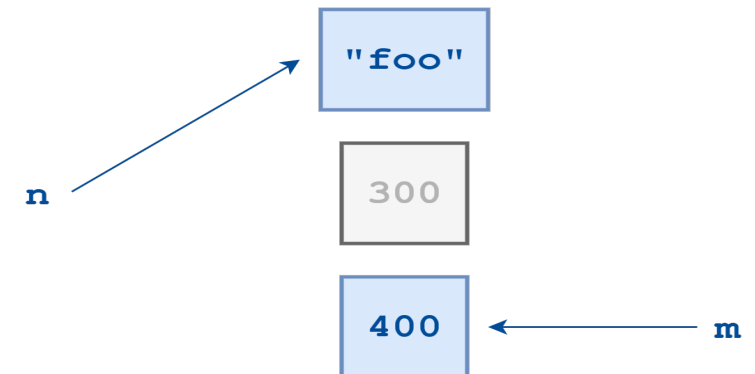


- **References to Separate Objects**

- Lastly, suppose this statement is executed next:

- `>>> n = "foo"`

- Now Python creates a string object with the value "foo" and makes n reference that.



- **Orphaned Object**

- There is no longer any reference to the integer object 300. It is orphaned, and there is no way to access it.

Created by Dr. Ritu Jain

- When the number of references to an object drops to zero, it is no longer accessible. At that point, its lifetime is over. Python garbage collector will eventually notice that it is inaccessible and reclaim the allocated memory so it can be used for something else.

Multiple Assignment

- Python allows multiple assignment. The statement
`x, y = 2, 3` binds `x` to 2 and `y` to 3.
- All of the expressions on the right-hand side of the assignment are evaluated before any bindings are changed. This is convenient since it allows you to use multiple assignment to swap the bindings of two variables.
- For example, the code

```
x, y = 2, 3
x, y = y, x
print('x =', x)
print('y =', y)
```

```
[>>> x, y=3, 2
[>>> x, y=y, x
[>>> print(x, y)
2 3 _
```

will print

```
x = 3
y = 2
```

Created by Dr. Ritu Jain

Comments

- **Comment:** It is a text that is intended only for the human reader — it is completely ignored by the interpreter.
- **Single Line comments**
 - Comment starts with #
 - Eg, #Prog to print hello world

Comments (Cont'd)

- **Multi-Line Comments**
 - **Python does not really have a syntax for multi line comments.**
 - insert # for each line.
 - Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it.
 - Eg. `""" This is a multi line comment."""`
 - Memory will be allocated to these strings internally. If these strings are not assigned to any variable, then they are removed from memory by the garbage collector and hence these can be used as comments.
 - This type of commenting is not recommended as they internally occupy memory and would waste the time of interpreter since interpreter has to check them.

Basic Data Types

- Variables can store data of different types, and different types can do different things.
- In Python we are not required to specify the type explicitly. Based on value provided, the type will be assigned automatically. Hence Python is **Dynamically Typed Language**.

Basic Data Types

Basic Data Types	Representation	Values	Examples
Integer	int	Set of all integers	i= 10
Float	float	Real numbers	j= 10.5 ej= 5e3
Complex	complex	Complex numbers	5 +4j
Boolean	bool	True and False	k= True
String	str	All characters enclosed in single or double quotes	l="True" m= 'Hello'

Created by Dr. Ritu Jain

Fundamental Data types are Immutable

- All Fundamental Data types are immutable. i.e. once we create an object, we cannot perform any changes in that object. If we are trying to change then with those changes a new object will be created. This non-changeable behavior is called immutability.
- In Python if a new object is required, then PVM won't create object immediately. First it will check if any object is available with the required content or not. If available then existing object will be reused. If it is not available then only a new object will be created.
- The advantage of this approach is memory utilization and performance will be improved. But the problem in this approach is several references pointing to the same object, by using one reference if we are allowed to change the content in the existing object then the remaining references will be affected. To prevent this immutability concept is required. [Created by Dr. Ritu Jain](#)
- According to this once we create an object we are not allowed to change content. If we are trying to change with those changes a new object will be created.

int Data Type

- whole number, positive or negative, without decimal point or fraction part.
- Eg. 200, -50, 0, 9896
- In python, **there is no limit for the size of int data type.**
- **It can store very large numbers very conveniently.**

[illegible]

```
<class 'int'>  
9999999999999999999999999999999999999999999999999999999
```

- **Note:**
 - In Python 2 we have long data type to represent very large integral values.
 - But in Python 3 there is no long type explicitly and we can represent long values also by using int type only.

int data type

- We can represent **int** values in the following ways:
 1. **Decimal form**
 2. **Binary form**
 3. **Octal form**
 4. **Hexadecimal form**

int data type

- **Decimal form(base-10):**

- It is the **default number system** in Python
- The allowed digits are: 0 to 9
- Eg: a =10, b=345

- **Binary form(Base-2)**

- The allowed digits are : 0 & 1
- Literal value should be prefixed with **0b** or **0B**
- **Correct:** 0b101, 0B111
- **Incorrect:** B101, 0b102

Created

```
num1= 0b101  
print(num1)
```

int data type

- **Octal Form(Base-8)**

- The allowed digits are : **0 to 7**
- Literal value should be prefixed with **zero followed by lowercase o or uppercase O (0o or 0O).**
- Correct: 0O123, 0o321
- Incorrect: O11, 0O458

```
num1 = 00112  
print(num1)
```

74

- **Hexa Decimal Form(Base-16):**

- The allowed digits are : **0 to 9, a-f** (both lower and upper cases are allowed)
- Literal value should be prefixed with **0x** or **0X**
- Eg:
 - a = 0XFACE
 - a = 0XBeef
 - a = 0XBeer

Created by Dr. Ritu Jain

- **Note:** We can specify literal values in decimal, binary, octal and hexa decimal forms. But PVM will always provide values only in decimal form.

Base Conversions

- Python provide the following in-built functions for base conversions
 - **bin():** We can use **bin()** to convert from any base to binary
 - bin() method converts and returns the binary equivalent string of a given integer.
 - Eg:

```
>>> bin(15)
'0b1111'
>>> bin(0o11)
'0b1001'
>>> bin(0X10)
'0b10000'
```

Base Conversions

- **oct(): We can use oct() to convert from any base to octal**

```
>>> oct(10)           Output: '0o12'  
>>> oct(0B1111)       Output: '0o17'  
>>> oct(0X123)         Output: '0o443'
```

- **hex(): We can use hex() to convert from any base to hexadecimal**

Eg:

```
>>> hex(16)           Output: '0x10'  
>>> hex(0B111111)     Output: '0x3f'  
>>> hex(0o77)         Output: '0x3f' Created by Dr. Ritu Jain
```


int()

- int() can be used to convert from any base to int.

```
>>> n1=0b11
>>> n2 =int(n1)
>>> print(n2, type(n2))
3 <class 'int'>
```

```
>>> n_oct=0o17
>>> n2_oct =int(n_oct)
>>> print(n2_oct, type(n2_oct))
15 <class 'int'>
```

- **int(str_var, base):** function to convert a string into integer.
- **str_var:** represents string format of a number
- **base:** base of the number system to be used for the string.

```
str=input("Enter hexadecimal number ")  
n=int(str, 16)  
print(n)
```

```
Enter hexadecimal number 0x11  
17
```

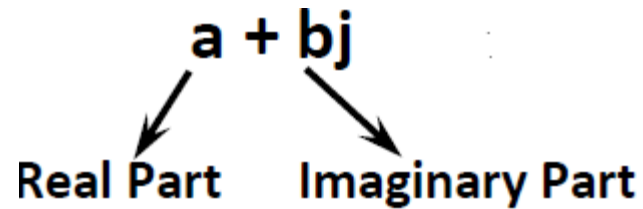
Created by Dr. Ritu Jain

float data type

- Floating point number that **contain a decimal point**.
- Eg num =55.56
- Floating point number **can also be written in exponential form (scientific notation)** where we use **e** or **E** to represent **power of 10**.
- Eg: f=**1.2e3**
- **print(f) output: 1200.0**
- The main advantage of exponential form is we can represent **big values in less memory**.
- **We can represent int values in decimal, binary, octal and hexadecimal forms. But we can represent float values only by using decimal form.**

Complex Data Type

- complex number is a number that is written in the form of



- **a and b can contain integers or floating point values.**
- **J can be lower case or uppercase (i.e $a+bj$ or $a+bJ$)**
- Eg:
 - $3+5j$
 - $10+5.5j$
 - $0.5+0.1j$

Created by Dr. Ritu Jain

Complex Data Type

- Note: **Complex data type** has some inbuilt attributes to retrieve the real part and **imaginary part**
- `c=10.5+3.6j`
- `c.real==>10.5`
- `c.imag==>3.6`
- We can use complex type generally in scientific applications and electrical engineering applications.

Data types

- **Identifying Data Type of Object**

- Syntax : `type(object)`
- Eg. `type(i)`

- **Type Conversion**

- **Syntax:** `datatype(object)`
- Eg. `Height = 165.5`

`Ht = int(Height)`

`type(Height)`

`type (Ht)`

Created by Dr. Ritu Jain

Type Coercion

- We can convert one type value to another type. This conversion is called Typecasting or Type coercion.
- The following are various inbuilt functions for type casting.
 1. int()
 2. float()
 3. complex()
 4. bool()
 5. str()
- **We can convert from any type to int except complex type.**
- **If we want to convert str type to int type, compulsory str should contain only integral value**
- **complex(x) is used to convert x into a complex number with real part x and imaginary part zero.**
- **complex(x, y) is used to convert x and y into a complex number such that:**
 - **x will be real part and y as imaginary part.**

- Example: **We can convert from any type to int except complex type.**

In [2]:

```
▶ c=3+2j  
print(int(c))
```

TypeError

Traceback (most recent call last)

<ipython-input-2-3724b731b4fc> in <module>

1 c=3+2j

----> 2 print(int(c))

TypeError: can't convert complex to int

Created by Dr. Ritu Jain

Type Coercion

- Few coercions are accepted, but not all types of coercions are possible.

- Eg1. `x = '2'`
 `x= int(x)`
 `type(x)`

- Eg2. `x="h"`
 `type(x)`
 `x= int(x) #valueError`

The int function can take a floating point number or a string, and turn it into an int. For floating point numbers, it *discards* the decimal portion of the number — a process we call *truncation towards zero* on the number line.

```
int(20.7)
```

Created by Dr. Ritu Jain

Type Casting: int

```
>>> int(123.987)
123
>>> int(10+5j)
TypeError: can't convert complex to int
>>> int(True)
1
>>> int(False)
0
>>> int("10")
10
>>> int("10.5")
ValueError: invalid literal for int() with base 10: '10.5'
>>> int("ten")
ValueError: invalid literal for int() with base 10: 'ten'
>>> int("0B1111")
ValueError: invalid literal for int() with base 10: '0B1111'
```

Type Coercion

```
>>> x=20
>>> type(x)
<class 'int'>
>>> bool(x)
True
>>> float(x)
20.0
>>> str(x)
'20'
>>> type(x)
<class 'int'>
>>> y=str(x)
>>> type(y)
<class 'str'>
>>> complex(x)
(20+0j)
```

Created by Dr. Ritu Jain

bool data type

- We can use this data type to represent Boolean values.
- The only allowed values for this data type are: **True and False**
- **Internally Python represents True as 1 and False as 0**
- Eg1:
 - `b=True`
 - `type(b)` **Output:** bool
- Eg2:
 - `a=10`
 - `b=20`
 - `c=a<b`
 - `print(c)` **Output:** True
- `True+True` **Output:** 2
- `True-False` **Output:** 1
- Any string is True, except empty strings.

Created by Dr. Ritu Jain

Type Casting: Bool

- 1) `bool(0)`==>False
- 2) `bool(1)`==>True
- 3) `bool(20)`==>True
- 4) `bool(13.5)`==>True
- 5) `bool(0.178)`==>True
- 6) `bool(0.0)`==>False
- 7) `bool(10-2j)`==>True
- 8) `bool(0+1.5j)`==>True
- 9) `bool(0+0j)`==>False
- 10) `bool("True")`==>True
- 11) `bool("False")`==>True
- 12) `bool("")`==>False
- 13) `bool("0")` ==> True

Created by Dr. Ritu Jain

Type Casting: float

We can use float() function to convert other type values to float type.

```
>>> float(10)
>>> float(10+5j)      #
>>> float(True)
>>> float(False)
>>> float("10")
>>> float("10.5")
>>> float("ten")      #
>>> float("0B1111")  #
```

Note:

1. We can convert any type value to float type except complex type.
2. Whenever we are trying to convert str type to float type compulsory str should be either integral or floating point literal and should be specified only in base-10.

Type Casting: complex()

- We can use complex() function to convert other types to complex type.
- **complex(x):** We can use this function to convert x into complex number with real part x and imaginary part 0.

Eg:

```
complex(10)
complex(10.5)
complex(True)
complex(False)
complex("10")
complex("10.5")
complex("ten")    #
```

- **complex(x,y):** We can use this method to convert x and y into complex number such that x will be real part and y will be imaginary part.
- Eg: `complex(10,-2)`
 `complex(True,False)`

Created by Dr. Ritu Jain

Examples:type casting Complex

```
print(complex(10))      #10+0j
print(complex(10.5))    #10.5+0j
print(complex(True))    #1+0j
print(complex(False))   #0j
print(complex("10"))    #10+0j
print(complex("10.5"))  #10.5+0j
#complex("ten") #ValueError: complex() arg is a malformed string
#print(complex("10.5","5.5")) # TypeError: complex() can't take second arg if first is a string
print(complex(True, True))
print(complex(False, True))
print(complex(True, False))
print(complex(False, False))
```

Created by Dr. Ritu Jain

String

- str represents String data type.
- A String is a sequence of characters enclosed within single quotes or double quotes.
- Triple quotes is used for multiline strings.
- Eg:

“””

Python is an
General purpose programming language

“””

- We can also use triple quotes to use single quote or double quote in our String.
- `" This is " character"`
- `' This i " Character '`
- We can embed one string in another string
- Eg. `“”You know that “String is a basic datatype”`

Created by Dr. Ritu Jain

```
>>> """You know, "String is data type"""
SyntaxError: EOL while scanning string literal
>>> """You know, "String is data type" """
'You know, "String is data type" '
```

String

- `s1 = "You're looking great today"`
- `s2 = 'You're looking great today'`
- `s3 = "You\"re looking great today"`
- `s4= ='You're looking great today'` **Syntax Error !**
- **In Python, we can represent char values also by using str type and explicitly char type is not available.**
- A string in Python can contain as many characters as you wish. The only limit is your machine's memory resources. A string can also be empty.

Created by Dr. Ritu Jain

String

- In Python, we can represent char values also by using str type and explicitly char type is not available.
- Eg:
 - 1) `>>> c='a'`
 - 2) `>>> type(c)`
 - 3) `<class 'str'>`

String

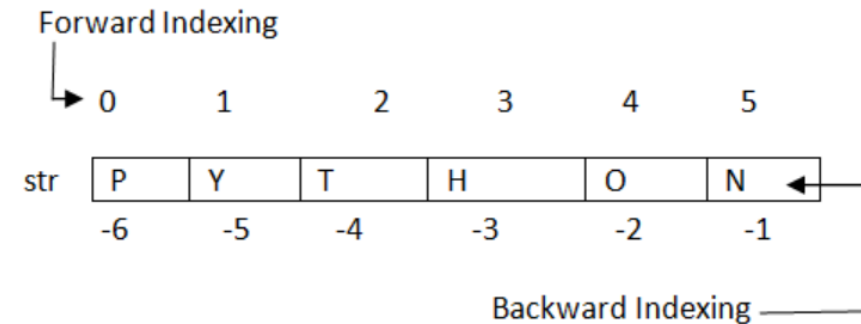
- Slice means a piece
- In Python Strings follows zero based index, i.e., index starts from 0
- Slicing operator ([], [:]): retrieve a subset of string

- Syntax: `str1[start : end <:step>]`
- Eg. `str1 = "HelloWorld"`

```
print(str1[1])  
print(str1[1:8:2])
```

Output: elWr

- The index can be either +ve or -ve.
- +ve index means forward direction from Left to Right
- -ve index means backward direction from Right to Left
- Concatenation operator (+)
 - `print("Hello" + "World")`
- Repetition operator (*)
 - `print("Hello"*3)`



Created by Dr. Ritu Jain

String

```
>>> s="Hello"  
>>> s[0]  
>>> s[1]  
>>> s[-1]  
>>> s[10]  
>>> s[1:40]  
>>> s[1:]  
>>> s[:4]  
>>> s[:]  
>>> s*3  
>>> len(s)
```

Created by Dr. Ritu Jain

String

- `title()` displays each word in titlecase, where each word begins with a capital letter.
- `upper()`: converts each letter into upper case
- `lower()`: converts each letter into lower case
- Eg. `name = "ada lovelace"`
 `print(name.title())`
 `print(name.upper())`
 `print(name.lower())`

Stripping Whitespace

- Python can look for extra whitespace on the right and left sides of a string. To ensure that no whitespace exists at the right end of a string, use the `rstrip()` method.
- **Eg1.** `favorite_language = ' python '`
 `favorite_language.rstrip()`
- **Eg2.** `favorite_language.lstrip()`
- **Eg3.** `favorite_language.strip()`
- In the real world, these stripping functions are used most often to clean up user input before it's stored in a program.

String

- Check it!
 - age = 23
 - message = "Happy " + age + "rd Birthday!"
 - print(message)

Type casting:str()

str(): We can use this method to convert other type values to str type

Eg:

```
>>> str(10)
```

```
>>> str(10.5)
```

```
>>> str(10+5j)
```

```
>>> str(True)
```

str data type

- String are immutable and hence you can not change the individual letters of string.
- name='hello'
- name[0]='p' #TypeError: 'str' object does not support item assignment

List

- List is an ordered, mutable, heterogeneous collection of elements, where duplicates also allowed.
- Objects of a list are also called as elements or components.
- List represent a group of values as a single entity where
 - insertion order is preserved
 - duplicates are allowed.
 - heterogeneous objects are allowed
 - can add and remove elements from it
- Lists are one of Python's most powerful features.
- Eg: [11, 22, 33, 44]
 - ['abc', 10, 10.5]

Created by Dr. Ritu Jain

List

- Elements of a list need not be of same data type.
- It is enclosed between two square brackets []. Individual elements in the list are separated by commas.
- **Empty List:**
 - `ls = []`
- **Creating and printing a list**
 - `ls=[20,5.5,'abc',True,20]`
 - `print(list)`
 - `list=[10,20,30,40]`
 - `print(list[0])`
 - `print(list[-1])`
 - `print(list[1:3])`
 - `list[0]=100`
 - `print(list)`
 - `print(type(list))`

Created by Dr. Ritu Jain

Tuple

- an ordered, immutable, heterogenous collection of elements, where duplicates also allowed.
- **Immutable**: once created they can't be modified. So, when you want to have a list of items that cannot change in your program, use tuple.
- Enclosed between ()
- Creating tuples
 - `t1=(10,20,30,40)`
 - `print(t1)`
 - `print(type(t1))`
 - `t2=(10, 10.5, "abc",True)`
 - `print(t2)`
 - `print(t2[0])`
 - `print(t2[-1])`
 - `t2[0]=100`

Created by Dr. Ritu Jain

Set

- Set is an unordered collection of data types that is mutable, and has no duplicate elements.
- The order of elements in a set is undefined though it may consist of various elements.
- The type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.
- In sets:
 - insertion order is not preserved
 - duplicates are not allowed
 - heterogeneous objects are allowed
 - index concept is not applicable
 - It is mutable collection, but it cannot contain mutable elements
 - Growable in nature

Created by Dr. Ritu Jain

Set

- `s1={10,20,30}`
- `print(s1)`

- `s2={10,20,30,10}`
- `print(s2)`

- `s3={100,0.5,10,200,'abc'}`
- `print(s3)`
- `print(s3[0])`

- `s1={1,2,(3,4)}`
- `print(s1)`

- `s2={1,2,[3,4]}`
- `print(s2)`

Created by Dr. Ritu Jain

Dictionary

- Unordered collection of comma separated key-value pairs, within {}
- Each *key* is connected to a value, and you can use a key to access the value associated with that key.
- Mutable type
- Colon is used to separate key, value pair.
- Creating a dictionary:
 - Fuel_Type = {"Petrol": 1, "Diesel":2, "CNG":3}
- Keys are immutable, but values can be modified.
- Printing the dictionary
 - print(Fuel_type) # {'Petrol': 1, 'Diesel': 2, 'CNG': 3}
 - print(Fuel_Type["Petrol"]) #1

Data types

- List
- int

Name	Type	Description
Integers	int	Whole numbers, such as: 3 300 200
Floating point	float	Numbers with a decimal point: 2.3 4.6 100.0
Strings	str	Ordered sequence of characters: "hello" 'Sammy' "2000" "楽しい"
Lists	list	Ordered sequence of objects: [10,"hello",200.3]
Dictionaries	dict	Unordered Key:Value pairs: {"mykey": "value", "name": "Frankie"}
Tuples	tup	Ordered immutable sequence of objects: (10,"hello",200.3)
Sets	set	Unordered collection of unique objects: {"a","b"}
Booleans	bool	Logical value indicating True or False

- Cl

None

- None is used to define a null value, or no value at all.
- None is not the same as 0, False, or an empty string.
- None is a data type of its own (NoneType) and only None can be None.

- **Null Vs None in Python**

- **None** – None is an instance of the NoneType object type.
- **Null** – There is no null in Python, we can use None instead of using null values.
- Example: `print(type(None))`
 - `var1 = None`
 - `print(type(var1))`

Note: If a function does not return anything, it returns None in Python.

•

Constants

- Constants concept is not applicable in Python.
- But it is convention to use only uppercase characters if we don't want to change value.
- Eg. MAX=10
- It is just convention but we can change the value.

Escape characters

- `\n` New Line
- `\t` Horizontal tab
- `\'` Single quote
- `\"` Double quote
- `\\` back slash symbol

Operators

- Operator: Symbol that perform some operation on variables.
 - The most elementary goal of an operator is to be part of an expression.
Operators by themselves don't do much
- Operands: variables on which operations are performed.
- Types of operators: unary, binary, ternary
- Operators in python:
 1. Arithmetic
 2. Relational
 3. Logical
 4. Assignment
 5. Bitwise
 6. Special operators

Created by Dr. Ritu Jain

Arithmetic Operators

Arithmetic operators are those operators that allow you to perform *arithmetic operations* on numeric values

- Binary Operators:

Operator	Eg,	Result (if a= 10, b=3)
Addition (+)	a+b	13
Subtraction (-)	a-b	7
Multiplication (*)	a*b	30
Division (/) (return always float)	a/b	3.3333333333333335
Floor Division or Integer Division(//)	a//b	3
Modulus (%)	a%b	1
Exponent (**)	a**b	1000

- a and b both can be int, float, complex type
- Unary operators: +,-

Arithmetic Operators

a = 10.5

b=2

Output

a+b

12.5

a-b

8.5

a*b

21.0

a/b

5.25

a//b

5.0

Floor division always rounds down

a%b

0.5

a**b

110.25

#a and b both can be int, float type

Note: The result of integer division is always rounded to the nearest integer value that is less than the real (not rounded) result.

Created by Dr. Rita Jain

Arithmetic Operators

- Division operator (/) always performs floating point arithmetic. Hence it will always return float value.
 - $10/2 \Rightarrow 5.0$
 - $10.0/2 \Rightarrow 5.0$
- But Floor division (//) can perform both floating point and integral arithmetic. If arguments are int type then result is int type.
 - If at least one argument is float type then result is float type.
 - $10//2 \Rightarrow 5$
 - $10.0//2 \Rightarrow 5.0$

Created by Dr. Ritu Jain

ZeroDivisionError

- Note: For any number x, $x/0$ and $x\%0$, $x//0$ always raises "ZeroDivisionError"

```
▶ 10/0
```

```
-----  
ZeroDivisionError                                Traceback (most recent call last)  
<ipython-input-4-e574edb36883> in <module>  
----> 1 10/0  
  
ZeroDivisionError: division by zero
```

```
▶ 10.0/0
```

```
-----  
ZeroDivisionError                                Traceback (most recent call last)  
<ipython-input-3-e796260e48b5> in <module>  
----> 1 10.0/0  
  
ZeroDivisionError: float division by zero
```

Repetition Operator

- If we use * operator for str type then compulsory one argument should be int and other argument should be str type.
- 2*“Hello“ Correct
- “Hello”*2 Correct
- 2.5*“Hello” O/P: **TypeError**: can't multiply sequence by non-int of type 'float'
- “Hello”*“Hello“ O/P: **TypeError**: can't multiply sequence by non-int of type 'str'

Order of Precedence

Decreasing Order of Precedence	Operators
Parentheses	()
Exponent	**
Division, Multiplication, Modulus, Floor division	/, *, %, //
Addition and Subtraction	+, -

Created by Dr. Ritu Jain

Precedence of Operators

- `x=1 + 2 ** 3 / 4 * 5`
- `print(2*'Yes' + 3* '!')`
- `print(type(1+5))`
- `print(type(1+5.0))`
- `print(type(1+'5'))`
- `print(type('1'+'5'))`
- `print(2+3*5/5)`
- `print(3+2**2**3)`

Created by Dr. Ritu Jain

Relational Operators

- Used to test numerical equalities and inequalities.
- allow you to *compare* operands.
- These operators always return a Boolean value (True or False)

Operator	Meaning	Example x=10, y=5	Result
>	Greater than	x > y	True
<	Less than	x < y	False
==	Equal to	x == y	False
!=	Not equal to	x != y	True
>=	Greater than or equal to	x >= y	True
<=	Less than or equal to	x <= y	False

Created by Dr. Ritu Jain

Relational operators can work with int, float, string

For numeric values, values are compared after removing trailing zeros after decimal point. Ex: 4.0 and 4 will be treated as equal

Use only == and != with complex numbers

Relational Operators (cont'd)

Output

```
print(3>3.0)  
print(3>=3.0).  
print(10 >4)  
print(10.5 > 10)
```

```
False  
True  
True  
True
```

Relational Operators (cont'd)

- **Relational operators can be applied on str types also.**
- String comparison in Python takes place character by character.
 - That is, characters in the same positions are compared from both the strings. If the characters fulfill the given comparison condition, it moves to the characters in the next position. Otherwise, it merely returns False.
- The comparisons are case-sensitive, hence same letters in different letter cases(upper/lower) will be treated as separate characters
- If two characters are different, then their Unicode value is compared; the character with the smaller Unicode value is considered to be lower.
- Uppercase letters are considered less than small letters.

Created by Dr. Ritu Jain

Relational Operators (cont'd)

- **Relational operators on str types:**

- ```
a="Hello"
b="Hello"
print("a > b is ",a>b)
print("a >= b is ",a>=b)
print("a < b is ",a<b)
print("a == b is ",a==b)
print("a != b is ",a!=b)
```

```
a > b is False
a >= b is True
a < b is False
a == b is True
a != b is False
```

Created by Dr. Ritu Jain



# Relational Operators (cont'd)

- Equality operators (`==` , `!=`): We can apply these operators for any type even for incompatible types also
  - `10==20`                      `False`
  - `10!= 20`                      `True`
  - `"Hello"=="Hello"`        `True`
  - `10=="Hello"`                `False`
  - `10+1j==10+1j`              `True`
- Relational operator have lower precedence than arithmetic operators
  - `a+5 > c-2` corresponds to `(a+5) > (c-2)`

Created by Dr. Ritu Jain

# Chaining of Relational Operators

- Relational operators can be chained.
- If we get all True, then only result will be True, otherwise False
- Example:
  - $x=15$
  - $10 < x < 20$
  - $10 \geq x < 20$
  - $10 < x > 20$
  - $4 > 2 \geq 2 > 1$

# Relational Operators (cont'd)

- Chaining concept is applicable for equality operators.
- If at least one comparison returns False then the result is False. Otherwise the result is True.
- $10 == 20 == 30 == 40$  False
- $10 == 10 == 10 == 10$  True

# Logical Operators

- Logical operators can have either of the two types of operands:
  - relational expressions (evaluates to either True or False) or
  - numbers or strings or lists (non boolean values)

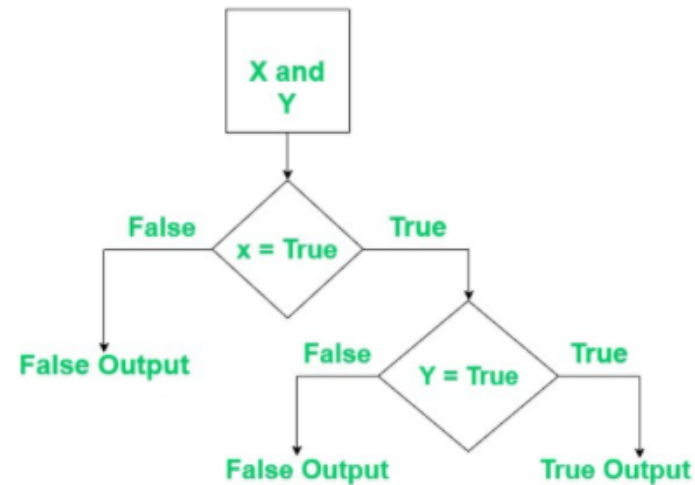
| Operator | Meaning                                            | Example: x =10, y=20, z =5 |
|----------|----------------------------------------------------|----------------------------|
| and      | True if both the operands are true                 | ((x<y) and (x<z))          |
| or       | True if either of the operands is true             | ((x<y) or (x<z))           |
| not      | True if operand is false (complements the operand) | not (x<y)                  |

# Logical Operators

- **For relational expression (boolean ) as operands:**
  - and ==> If both arguments are True then only result is True
  - or ==> If atleast one arugemnt is True then result is True
  - not ==> complement
  - Eg:
    - True and False ==> False
    - True or False ==> True
    - not False ==> True
- **Note:** If the first expression evaluated to be false while using and operator, then the further expressions are not evaluated.
- **Note:** If the first expression evaluated to be True while using or operator, then the further expressions are not evaluated. Created by Dr. Ritu Jain
- In the case of multiple operators, Python always evaluates the expression from left to right.

# Logical and operator

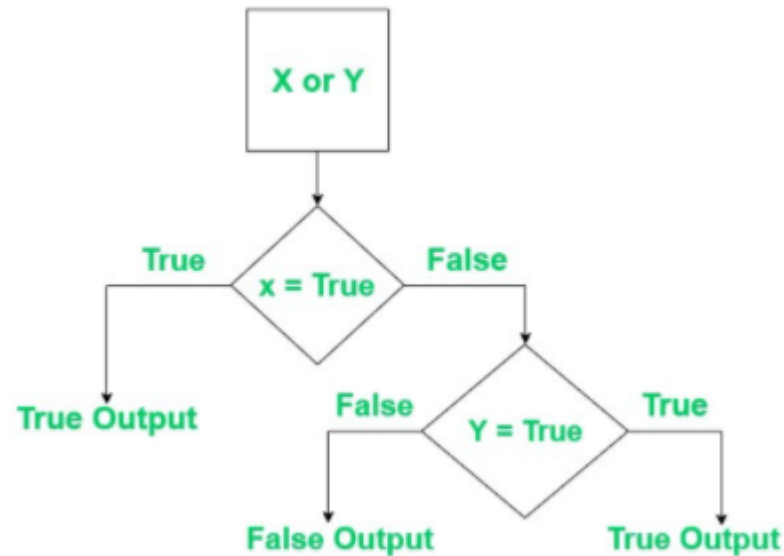
- Logical and operator returns True if both the operands are True else it returns False.



- Note:** If the first expression evaluated to be false while using and operator, then the further expressions are not evaluated.

# Logical or operator

- Logical or operator returns True if either of the operands is True.

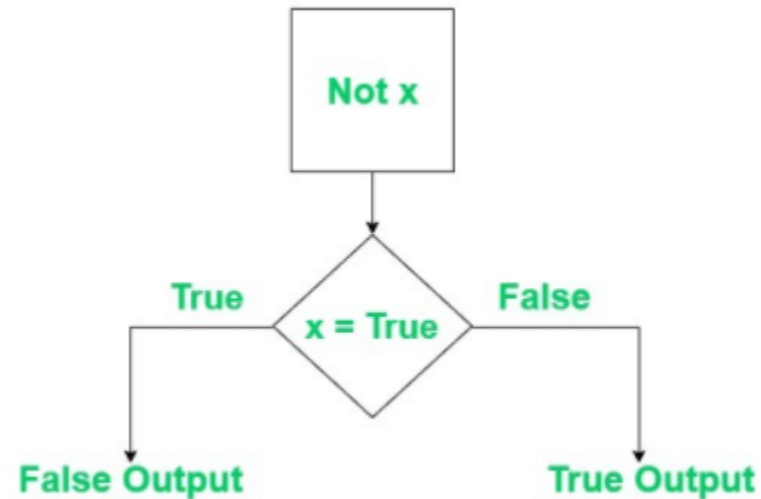


- **Note:** If the first expression evaluated to be True while using or operator, then the further expressions are not evaluated.

Created by Dr. Ritu Jain

# Logical not operator

- Logical not operator work with the single boolean value. If the boolean value is True it returns False and vice-versa.



Created by Dr. Ritu Jain



# Logical Operators

- **When evaluated with logical operators, all the following are considered false:**
  - Boolean value False
  - Any value that is numerically zero (0, 0.0, 0.0+0.0j)
  - An empty string
  - An object of a built-in composite data type such as list, tuple, dict, and set which is empty
  - The special value denoted by the Python keyword None
- Virtually any other object built into Python is regarded as true.

# Logical Operators

- **Rules for non-boolean types when evaluated with logical operators:**
  - **x and y:** if x evaluates to false return x otherwise return y. If first argument is zero then result is zero otherwise result is y
  - Eg:
    - $10 \text{ and } 20 \implies 20$
    - $0 \text{ and } 20 \implies 0$
    - $20 < 10 \text{ and } "a" + 1 > 1 \implies$  it will ignore second operand, if first operand is false, even if second operand is syntactically wrong
  - **x or y:** If x evaluates to True then result is x otherwise result is y
    - $10 \text{ or } 20 \implies 10$
    - $0 \text{ or } 20 \implies 20$
    - $20 > 10 \text{ or } "a" + 1 > 1 \implies$  it will ignore second operand, if first operand is true, even if second operand is syntactically wrong
  - **not x:** If x evaluates to False then result is True otherwise False
    - $\text{not } 10 \implies \text{False}$
    - $\text{not } 0 \implies \text{True}$

Created by Dr. Ritu Jain

# Logical Operators

- **Eg:**
  - **“Hello” and “Hello World” ==>”Hello World”**
  - **“” and “Hello” ==>””**
  - **“Hello” and “” ==>””**
  - **“” or “Hello” ==>“Hello”**
  - **“Hello” or “”==>”Hello”**
  - **not”” ==>True**
  - **not “Hello” ==>False**

# Assignment Operators

also know

| Operator | Example | Equivalent to |
|----------|---------|---------------|
| +=       | a+=b    | a= a+b        |
| -=       | a-=b    | a=a-b         |
| *=       | a*=b    | a=a*b         |
| /=       | a/=b    | a=a/b         |
| //=      | a//=b   | a=a//b        |
| **=      | a**=b   | a=a**b        |
| &=       | a&=b    | a =a&b        |
| =        | a =b    | a= a b        |
| ^=       | a^=b    | a=a^b         |
| >>=      | a>>=b   | a=a>>b        |
| <<=      | a<<=b   | a=a<<b        |

# Multiple Assignment

- `x=y=z=10`
- Note: While assigning values through multiple assignment, Python first evaluate the RHS expressions and then assigns them to LHS.
  - `a,b,c=1,2,3`
  - `a,b,c=a+1,b+2,c+3`
- Expressions on RHS are evaluated from left to right.
  - `x,x=5,10`
  - `y,y=x+10,x+20`
  - `print(x,y)`

# Ternary Operator

- **x = firstValue if condition else secondValue**
- If condition is True then firstValue will be considered else secondValue will be considered.
- Eg:

```
#WAP to find greater mong two numbers
n1,n2=10,20
g=n1 if n1>n2 else n2
print("Greater numbers among ",n1, "and ",n2, " is ",g)
```

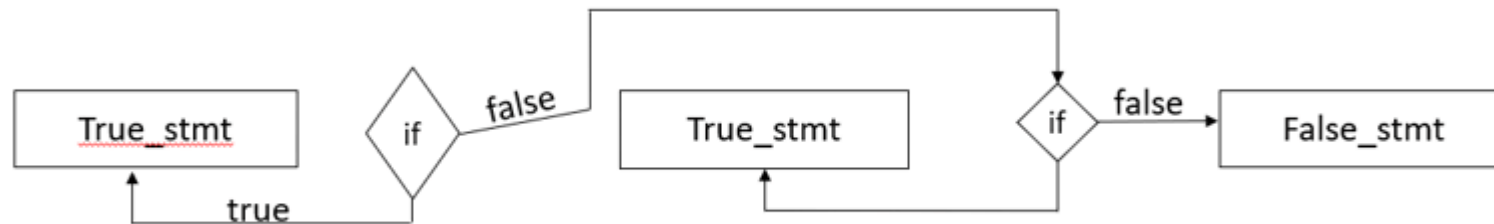
✓ 0.0s

Greater of 10 and 20 is 20

Created by Dr. Ritu Jain

# Nested Ternary Operator

- In the below code, the compiler first checks the if condition, if the condition is true then it prints the true statement, if the condition is false then it checks for the next if condition in else block and prints true statement if the condition is true otherwise prints the false statement.



Created by Dr. Ritu Jain

# Nested Ternary Operator

```
#WAP to find greatest among three numbers
n1,n2,n3=10,20,30
g=n1 if (n1>n2 and n1>n3) else (n2 if (n2>n3) else n3)
print("Greatest of ",n1, "and ",n2, " and ",n3, " is ",g)
```

✓ 0.0s

Greatest of 10 and 20 and 30 is 30

•

Created by Dr. Ritu Jain



# BITWISE OPERATORS

- We can apply these operators bitwise. (Operate **bit** by bit.)

- Integers are treated as string of binary digits

| Operator          | Description                                                     | Example: x=10, y=12<br>In binary x=1010<br>y=1100 |
|-------------------|-----------------------------------------------------------------|---------------------------------------------------|
| & Bitwise AND     | Operator copies bit if it exists in both operand.               | x & y results 1000                                |
| Bitwise OR        | Operator copies bit if it exists in either operand.             | x y results 1110                                  |
| ^ Bitwise XOR     | Operator copies bit if it exists only in one operand.           | x^y results 0110                                  |
| ~ Bitwise inverse | Unary operator, Operand is used to opposite the bits of operand | ~x results -1011                                  |
| << Left Shift     | Shift the bits towards left                                     | x<< 2 results 101000                              |
| >> Right shift    | Shift the bits towards right                                    | x>>2 results 0010                                 |

# Special operators

Python defines the following 2 special operators

1. Identity Operators
2. Membership operators

# Object Identity

- In Python, every object that is created is given a number that uniquely identifies it. It is guaranteed that no two objects will have the same identifier during any period in which their lifetimes overlap.
- Once an object's reference count drops to zero and it is garbage collected, then its identifying number becomes available and may be used again.
- The built-in Python function **id()** returns an object's integer identifier.
- Using the `id()` function, you can verify that two variables indeed point to the same object.

# Object Identity

```
>>> n = 300
```

```
>>> m = n
```

```
>>> id(n)
```

```
60127840
```

```
>>> id(m)
```

```
60127840
```

```
>>> m = 400
```

```
>>> id(m)
```

```
60127872
```

Created by Dr. Ritu Jain

# Identity Operators

- We can use identity operators (**is**, **is not**) for address comparison.
  - **is**
    - r1 is r2 returns True if both r1 and r2 are pointing to the same object
  - **is not**
    - r1 is not r2 returns True if both r1 and r2 are not pointing to the same object
- **NOTE: Use is operator for address comparison where as == operator for content comparison.**

# Membership operators

- We can use Membership operators to check whether the given object present in the
- given collection.(It may be String,List,Set,Tuple or Dict)
- **in**
  - Returns True if the given object present in the specified Collection
- **not in**
  - Retrurns True if the given object not present in the specified Collection



| Operator                                         | Description                                                                                                                                                                                               | Associativity |
|--------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| ( )                                              | Parentheses                                                                                                                                                                                               | left-to-right |
| **                                               | Exponent                                                                                                                                                                                                  | right-to-left |
| +, -, ~                                          | Unary plus, unary minus, bitwise Not                                                                                                                                                                      | left-to-right |
| * / %                                            | Multiplication/division/modulus                                                                                                                                                                           | left-to-right |
| + -                                              | Addition/subtraction                                                                                                                                                                                      | left-to-right |
| << >>                                            | Bitwise shift left, Bitwise shift right                                                                                                                                                                   | left-to-right |
| < <=<br>> >=                                     | Relational less than/less than or equal to<br>Relational greater than/greater than or equal to                                                                                                            | left-to-right |
| == !=                                            | Relational is equal to/is not equal to                                                                                                                                                                    | left-to-right |
| is, is not<br>in, not in                         | Identity<br>Membership operators                                                                                                                                                                          | left-to-right |
| &                                                | Bitwise AND                                                                                                                                                                                               | left-to-right |
| ^                                                | Bitwise exclusive OR                                                                                                                                                                                      | left-to-right |
|                                                  | Bitwise inclusive OR                                                                                                                                                                                      | left-to-right |
| not                                              | Logical NOT                                                                                                                                                                                               | right-to-left |
| and                                              | Logical AND                                                                                                                                                                                               | left-to-right |
| or                                               | Logical OR                                                                                                                                                                                                | left-to-right |
| =<br>+= -=<br>*= /=<br>%= &=<br>^=  =<br><<= >>= | Assignment<br>Addition/subtraction assignment<br>Multiplication/division assignment<br>Modulus/bitwise AND assignment<br>Bitwise exclusive/inclusive OR assignment<br>Bitwise shift left/right assignment | right-to-left |

## Operator Precedence and Associativity

Created by Dr. Ritu Jain