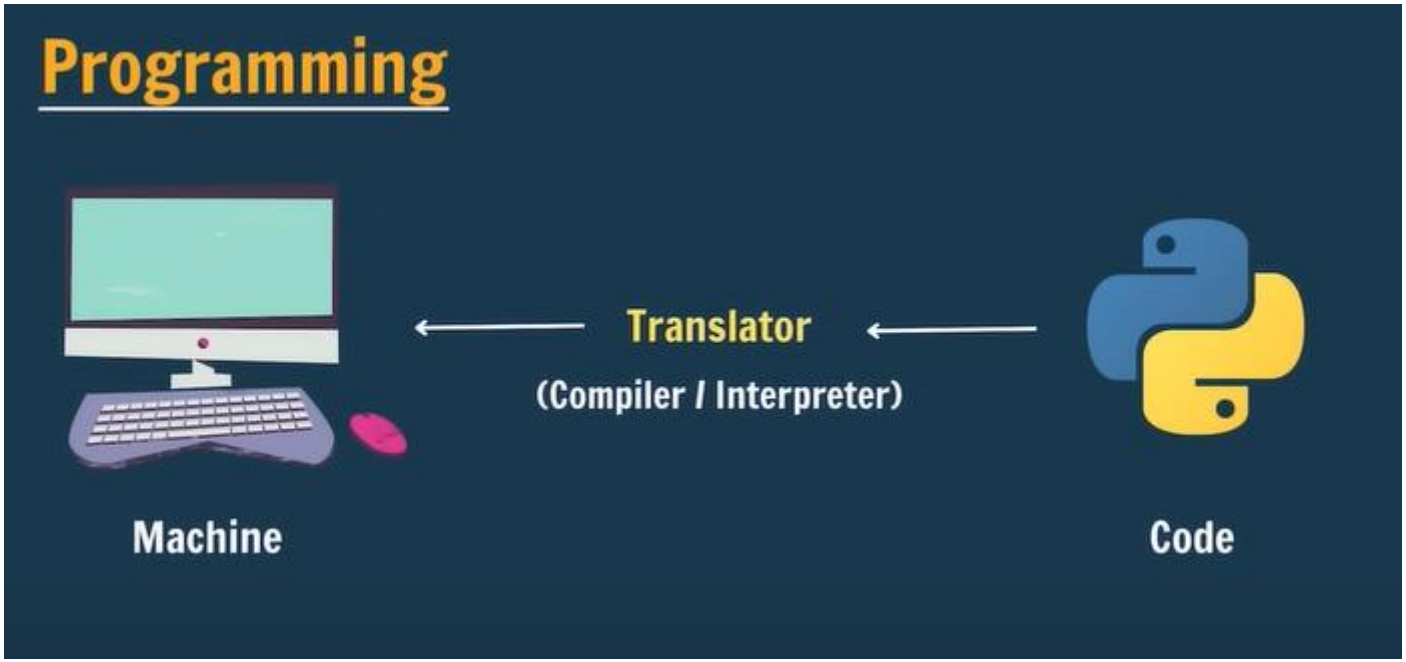
Presentation of Python For Beginner

Presented By : Deepak Kumar
(Application Architecture)

WHAT IS PROGRAM?

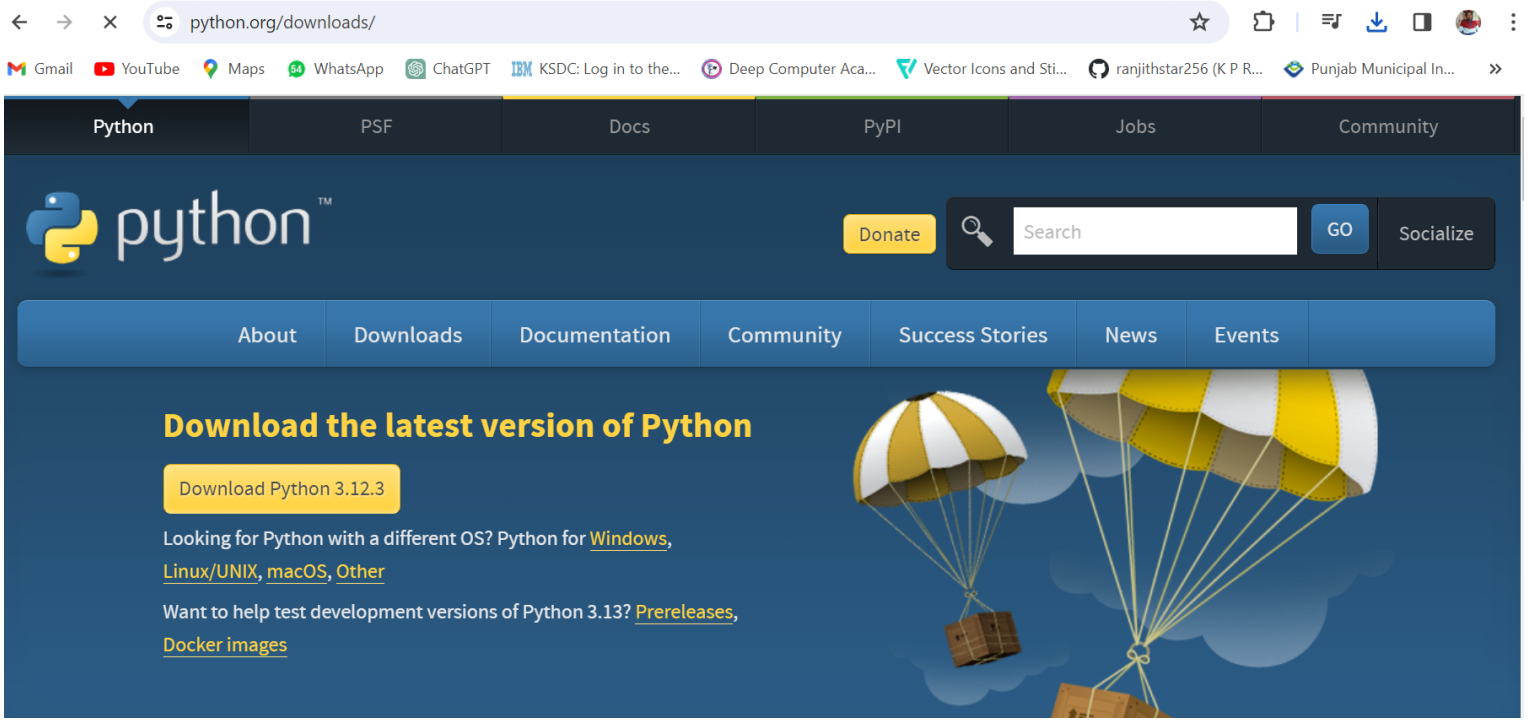
- An ordered set of instructions to be executed by a computer to carry out a specific task is called a program, and the language used to specify this set of instructions to the computer is called a programming language.
- As we know that computers understand the language of 0s and 1s which is called machine language or low level language. However, it is difficult for humans to write or comprehend instructions using 0s and 1s. This led to the advent of high-level programming languages like Python, C++, Visual Basic, PHP, Java that are easier to manage by humans but are not directly understood by the computer.



What is Python?

Python is simple & easy
Free & Open Source
High Level Language
Developed by Guido van Rossum
Portable

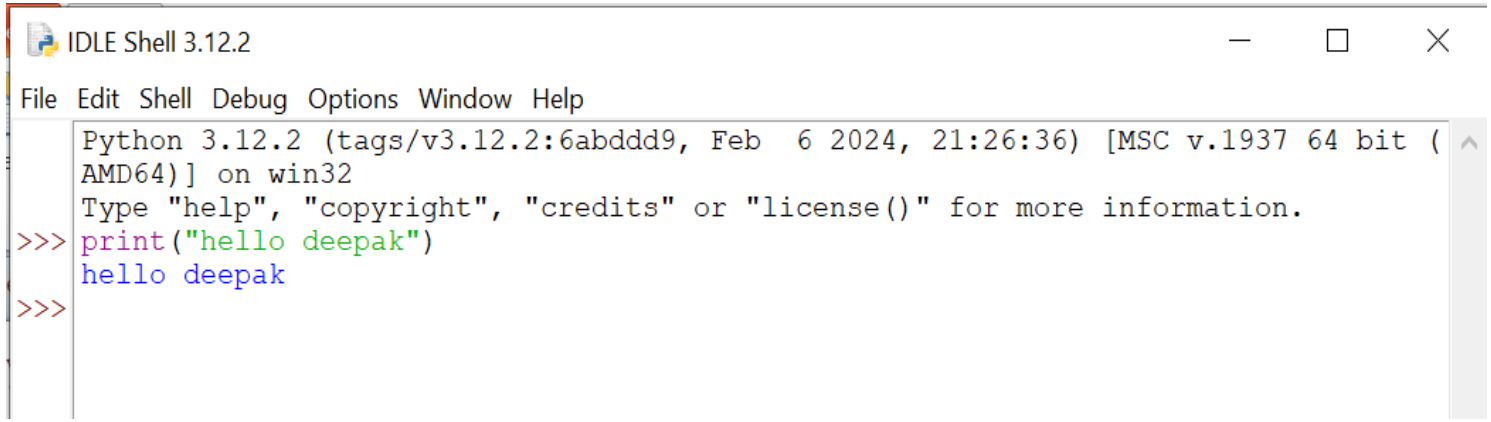
INSTALLATION OF PYTHON



EXECUTION MODES

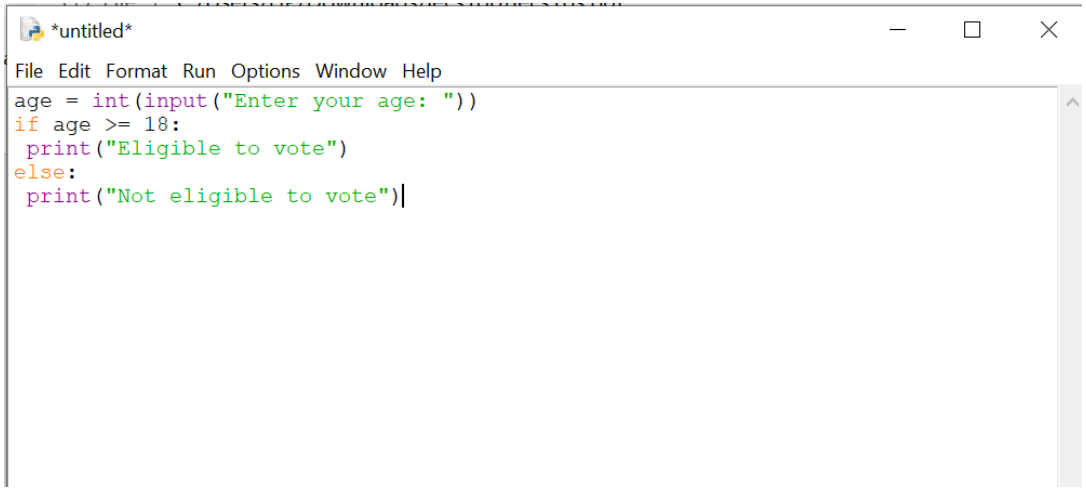
- There are two ways to use the Python interpreter:
- a) Interactive mode

To work in the interactive mode, we can simply type a Python statement on the >>> prompt directly.



B) SCRIPT MODE

- In the script mode, we can write a Python program in a file, save it and then use the interpreter to execute it. Python scripts are saved as files where file name has extension “.py”.



PYTHON KEYWORDS

- Keywords are reserved words. Each keyword has a specific meaning to the Python interpreter, and we can use a keyword in our program only for the purpose for which it has been defined. As Python is case sensitive, keywords must be written exactly as they are available.

Examples of Python Keywords

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

VARIABLES

- A variable in a program is uniquely identified by a name (identifier). Variable in Python refers to an object — an item or element that is stored in the memory. Value of a variable can be a string.

(e.g., ‘b’, ‘Global Citizen’), numeric (e.g., 345) or any combination of alphanumeric characters (CD67).

gender = 'M'

message = "Keep Smiling"

price = 987.9

WRITE A PYTHON PROGRAM TO FIND THE AREA OF A RECTANGLE GIVEN THAT ITS LENGTH IS 10 UNITS AND BREADTH IS 20 UNITS

- #To find the area of a rectangle

- length = 10
- breadth = 20
- area = length * breadth
- print(area)

Output: 200

COMMENTS

- Comments are used to add a remark or a note in the source code. Comments are not executed by interpreter.

They are added with the purpose of making the source code easier for humans to understand.

In Python, a comment starts with # (hash sign). Everything following the # till the end of that line is treated as a comment and the interpreter simply ignores it while executing the statement.

Example

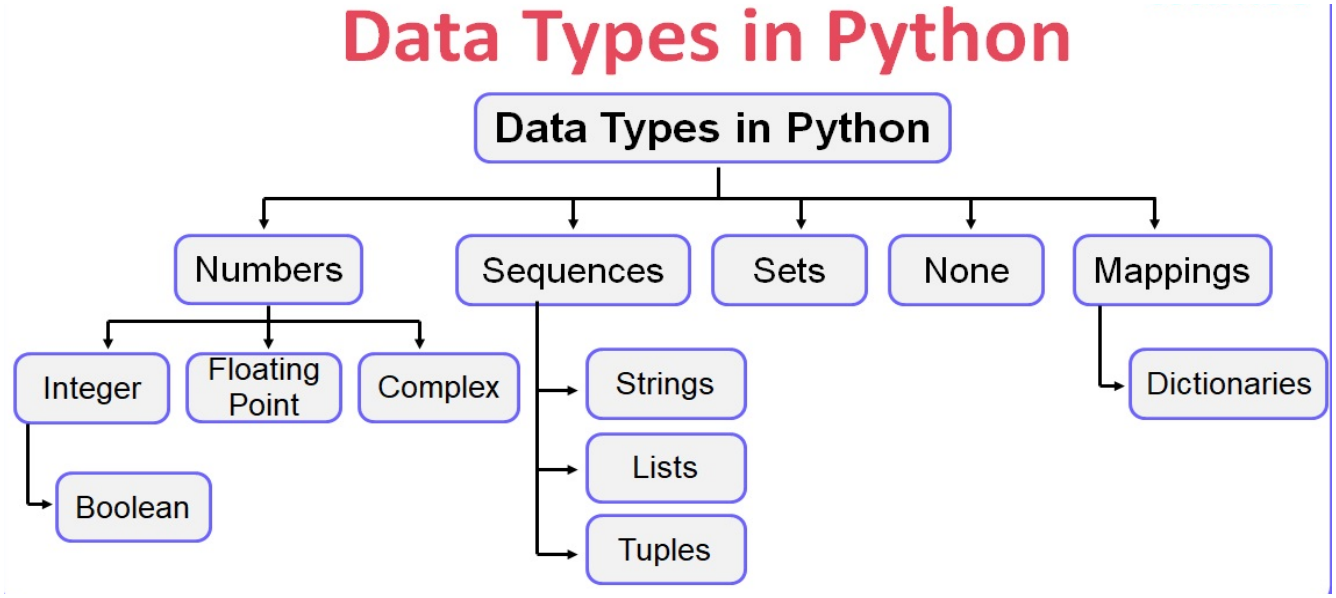
```
#Variable amount is the total spending on
#grocery
```

EVERYTHING IS AN OBJECT

- Python treats every value or data item whether numeric, string, or other type (discussed in the next section) as an object in the sense that it can be assigned to some variable or can be passed to a function as an argument.

DATA TYPES

- Every value belongs to a specific data type in Python. Data type identifies the type of data values a variable can hold and the operations that can be performed on that data.



NUMBER

- Number data type stores numerical values only. It is further classified into three different types: int, float and complex.

Type/ Class	Description	Examples
int	integer numbers	–12, –3, 0, 125, 2
float	real or floating point numbers	–2.04, 4.0, 14.23
complex	complex numbers	3 + 4j, 2 – 2j

- Boolean data type (bool) is a subtype of integer. It is a unique data type, consisting of two constants, True and False. Boolean True value is non-zero, non-null and non-empty. Boolean False is the value zero.

SEQUENCE

- A Python sequence is an ordered collection of items, where each item is indexed by an integer. The three types of sequence data types available in Python are Strings, Lists and Tuples.
- **A) String**

String is a group of characters. These characters may be alphabets, digits or special characters including spaces. String values are enclosed either in single quotation marks (e.g., ‘Hello’) or in double quotation marks (e.g., “Hello”).

- **B) List**
- List is a sequence of items separated by commas and the items are enclosed in square brackets [ ].

#To create a list

list1 = [5, 3.4, "New Delhi", "20C", 45]

#print the elements of the list list1

print(list1) [5, 3.4, 'New Delhi', '20C', 45]

(C) Tuple

Tuple is a sequence of items separated by commas and items are enclosed in parenthesis ( ). This is unlike list, where values are enclosed in brackets [ ]. Once created, we cannot change the tuple.

#create a tuple tuple1

tuple1 = (10, 20, "Apple", 3.4, 'a')

#print the elements of the tuple tuple1

print(tuple1) (10, 20, "Apple", 3.4, 'a')

SET

- Set is an unordered collection of items separated by commas and the items are enclosed in curly brackets { }. A set is similar to list, except that it cannot have duplicate entries. Once created, elements of a set cannot be changed.

```
#create a set
set1 = {10,20,3.14,"New Delhi"}  print(type(set1))
print(set1)
 {10, 20, 3.14, "New Delhi"}
```

#duplicate elements are not included in set

```
set2 = {1,2,1,3}
print(set2)
 {1, 2, 3}
```

NONE

- None is a special data type with a single value. It is used to signify the absence of value in a situation. None supports no special operations, and it is neither same as False nor 0 (zero).

```
myVar = None
print(type(myVar))
print(myVar)
```

None

MAPPING

Mapping is an unordered data type in Python. Currently, there is only one standard mapping data type in Python called dictionary.

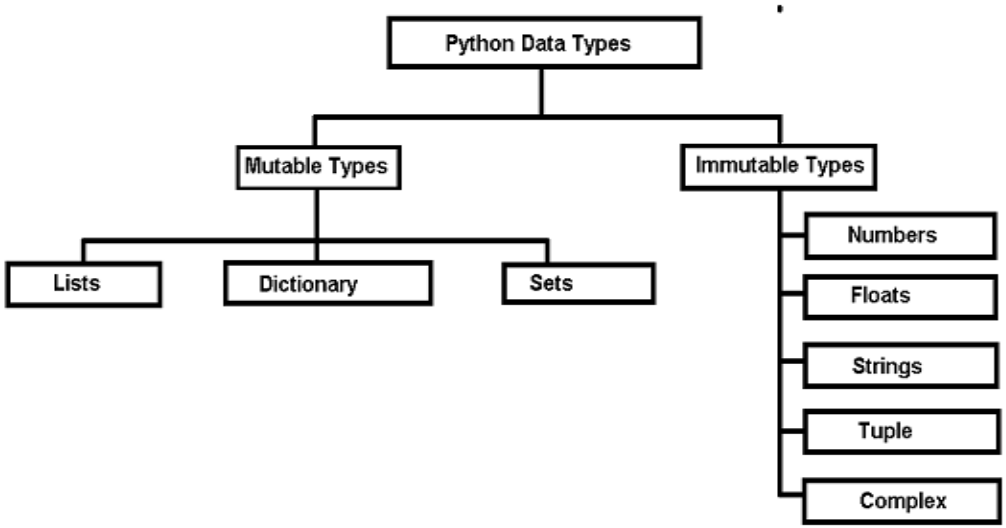
(A) Dictionary

Dictionary in Python holds data items in key-value pairs. Items in a dictionary are enclosed in curly brackets { }. Dictionaries permit faster access to data. Every key is separated from its value using a colon (:) sign. The key : value pairs of a dictionary can be accessed using the key.

```
#create a dictionary
dict1 = {'Fruit':'Apple', 'Climate':'Cold', 'Price(kg)':120} \
print(dict1)
{'Fruit': 'Apple', 'Climate': 'Cold', 'Price(kg)': 120}
print(dict1['Price(kg)'])
120
```

MUTABLE AND IMMUTABLE DATA TYPES

- Variables whose values can be changed after they are created and assigned are called **mutable**. Variables whose values cannot be changed after they are created and assigned are called immutable.



OPERATORS

- An operator is used to perform specific mathematical or logical operation on values. The values that the operators work on are called operands. For example, in the expression 10 + num, the value 10, and the variable num are operands and the + (plus) sign is an operator.

ARITHMETIC OPERATORS

- Python supports arithmetic operators that are used to perform the four basic arithmetic operations as well as modular division, floor division and exponentiation.

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 = 4 and 9.0//2.0 = 4.0

RELATIONAL OPERATORS

- Relational operator compares the values of the operands on its either side and determines the relationship among them. Assume the Python variables num1 = 10, num2 = 0, num3 = 10, str1 = "Good", str2 = "Afternoon" for the following examples.

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

ASSIGNMENT OPERATORS

- Assignment operator assigns or changes the value of the variable on its left.

Operator	Example	Equivalent Expression (m=15)	Result
=	y = a+b	y = 10 + 20	30
+=	m +=10	m = m+10	25
-=	m -=10	m = m-10	5
*=	m *=10	m = m*10	150
/=	m /=10	m = m/10	1.5
%=	m %=10	m = m%10	5
**=	m**=2	m = m**2 or $m = m^2$	225
//=	m//=10	m = m//10	1

LOGICAL OPERATORS

- There are three logical operators supported by Python. These operators **(and, or, not)** are to be written in lower case only.

Python - Logical Operators

- not

x	not x
False	True
True	False

- and

x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

- or

x	y	x or y
False	False	False
False	True	True
True	False	True
True	True	True

Operator Priority

IDENTITY OPERATORS

- Identity operators are used to determine whether the value of a variable is of a certain type or not. Identity operators can also be used to determine whether two variables are referring to the same object or not. There are two identity operators.

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

MEMBERSHIP OPERATORS

- Membership operators are used to check if a value is a member of the given sequence or not.

Membership Operators

Operator	Description	Example (Try in Lab)

in	Returns True if the variable/value is found in the specified sequence and False otherwise	>>> a = [1,2,3]
		>>> 2 in a
		True
		>>> '1' in a
		False

not in	Returns True if the variable/value is not found in the specified sequence and False otherwise	>>> a = [1,2,3]
		>>> 10 not in a
		True
		>>> 1 not in a
		False

EXPRESSIONS

An expression is defined as a combination of constants, variables, and operators. An expression always evaluates to a value. A value or a standalone variable is also considered as an expression but a standalone operator is not an expression.

Examples:

(i) 100 (iv) 3.0 + 3.14 (ii) num (v) 23/3 -5 * 7(14 -2) (iii) num – 20.4 (vi) "Global" + "Citizen"

STATEMENT

- In Python, a statement is a unit of code that the Python interpreter can execute.

Example

```
x = 4 #assignment statement
 cube = x ** 3 #assignment statement
 print (x, cube) #print statement
```

4 64

INPUT AND OUTPUT

- Sometimes, a program needs to interact with the user's to get some input data or information from the end user and process it to give the desired output. In Python, we have the input() function for taking the user input. The input() function prompts the user to enter data. It accepts all user input as string. The user may enter a number or a string but the input() function treats them as strings only. The syntax for input() is:

input ([Prompt])

INPUT

```
fname = input("Enter your first name: ")
 Enter your first name: Arnab
 age = input("Enter your age: ")
Enter your age: 19
type(age)
 <class 'string'>
```

#function int() to convert string to integer

```
age = int( input("Enter your age:"))
 Enter your age: 19
type(age)
<class 'int'>
```

TYPE CONVERSION

- As and when required, we can change the data type of a variable in Python from one type to another. Such data type conversion can happen in two ways: either explicitly (forced) when the programmer specifies for the interpreter to convert a data type to another type; or implicitly, when the interpreter understands such a need by itself and does the type conversion automatically.

Before not using type conversion

```
num1 = input("Enter a number and I'll double it: ")
num1 = num1 * 2 print(num1)
```

Enter a number and I'll double it: 2

22

After using type conversion

```
num1 = input("Enter a number and I'll double it: ")
num1 = int(num1)    #convert string input to integer
num1 = num1 * 2 print(num1)
```

Enter a number and I'll double it: 2

4

EXPLICIT CONVERSION

Explicit conversion, also called type casting happens when data type conversion takes place because the programmer forced it in the program. The general form of an explicit data type conversion is:

(new_data_type) (expression)

Function	Description
`int(x)`	Converts x to an integer
`float(x)`	Converts x to a floating-point number
`str(x)`	Converts x to a string representation
`chr(x)`	Converts ASCII value of x to character
`ord(x)`	returns the character associated with the ASCII code x

IMPLICIT CONVERSION

Implicit conversion, also known as coercion, happens when data type conversion is done automatically by Python and is not instructed by the programmer.

 # Program to show implicit conversion from int to float.

num1 = 10 #num1 is an integer

num2 = 20.0 #num2 is a float

sum1 = num1 + num2 #sum1 is sum of a float and an integer

print(sum1)

print(type(sum1))

Output: 30.0

DEBUGGING

The process of identifying and removing such mistakes, also known as bugs or errors, from a program is called debugging. Errors occurring in programs can be categorised as:

i) Syntax errors
ii) Logical errors
iii) Runtime errors

SYNTAX ERRORS

Like other programming languages, Python has its own rules that determine its syntax. The interpreter interprets the statements only if it is syntactically (as per the rules of Python) correct. If any syntax error is present, the interpreter shows error message(s) and stops the execution there. For example, parentheses must be in pairs, so the expression (10 + 12) is syntactically correct, whereas (7 + 11 is not due to absence of right parenthesis. Such errors need to be removed before the execution of the program.

LOGICAL ERRORS

A logical error is a bug in the program that causes it to behave incorrectly. A logical error produces an undesired output but without abrupt termination of the execution of the program.

For example, if we wish to find the average of two numbers 10 and 12 and we write the code as 10 + 12/2, it would run successfully and produce the result 16. Surely, 16 is not the average of 10 and 12. The correct code to find the average should have been (10 + 12)/2 to give the correct output as 11. Logical errors are also called semantic errors as they occur when the meaning of the program (its semantics) is not correct.

RUNTIME ERROR

- A runtime error causes abnormal termination of program while it is executing. Runtime error is when the statement is correct syntactically, but the interpreter cannot execute it. Runtime errors do not appear until after the program starts running or executing.

For example, we have a statement having division operation in the program. By mistake, if the denominator entered is zero then it will give a runtime error like “division by zero”.

```
#Runtime Errors Example
num1 = 10.0
num2 = int(input("num2 = "))
#if user inputs a string or a zero, it leads to runtime error
print(num1/num2)
```

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
======================= RESTART: C:\NCERT\prog5-11.py =======================
num2 = apple
Traceback (most recent call last):
  File "C:\NCERT\prog5-11.py", line 5, in <module>
    num2 = int(input("num2 = "))
ValueError: invalid literal for int() with base 10: 'apple'
>>>
```

Thank You