

Python



BRIDGE COURSE

TOPICS COVERED

95%

- Python Character Set
- Data Types
- Errors in Python Programs
- Python Tokens
- Find Data Type
- Some More Programs



PYTHON CHARACTER SET

A set of valid characters recognised by a language is called a character set. The characters used in the Python source program belong to the Unicode standard. The characters in Python are grouped into the following categories:

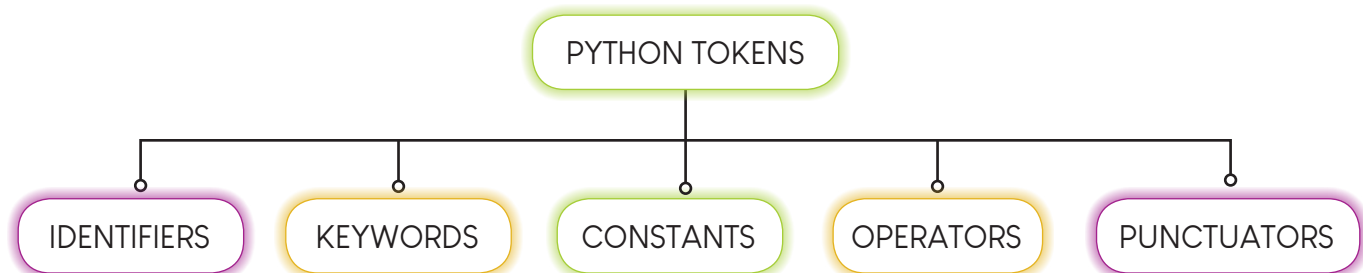
- Alphabet:** A to Z (uppercase), a to z (lowercase)
- Digits:** 0 to 9
- Special characters** (:), colon, (), +, -, *, /, ^, &, %, \$, {, }, [], !, _ (underscore), <, >, ?, @, , ; etc. (white space, blank space, horizontal tab (→), carriage return, new line, form feed)

Python can process any of the ASCII and Unicode characters as data or as literals.



PYTHON TOKENS

A token is the smallest element of a Python program that is meaningful to the interpreter. You have learned about some tokens in the previous class. Now learn more about them in detail.



IDENTIFIERS

An identifier is a sequence of characters taken from the Python character set. It refers to variables, functions and arrays. The rules for identifiers are:

- Only letters, digits and an underscore are permitted.
- Must start with a letter between A to Z or between a to z or an underscore (_).
- Uppercase and lowercase are distinct because Python is a case sensitive language.
- Special characters are not allowed.

Some examples of valid identifiers are Myvar, myvar_1, Sum_of_the_numbers, PASS and _Sum.

Some examples of invalid identifiers are False, Var^2, Var 1 and lvar.

KEYWORDS

Keywords are the reserved words. They are predefined words. Keywords cannot be used as an identifier.

Some commonly used keywords in Python are given below:

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

CONSTANTS OR LITERALS

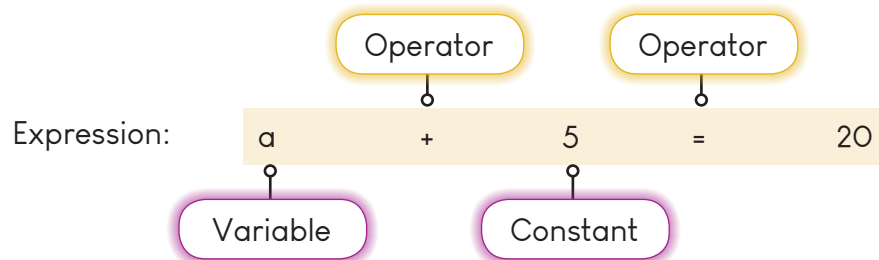
Constants are fixed values that do not change during the execution of a program. Literals are the type of constant. Literals refer to any number, text and other information that represents a value. Python supports the following literals:

Literals	Example
String literals	"hello", '12345'
Integer literals	0, 1, 2, -1, -2
Long literals	89675
Floating-point literals	3.14
Complex literals	12j
Boolean literals	True or False
Special literals	None
Unicode literals	u"hello"
List literals	[5,6,7]

OPERATORS

Operators are special symbols that are used to perform specific operation on operands and give a meaningful result. Operands are the data involved in the mathematical operation, and may be stored in variables and constants.

Combination of variables, constants and operators makes an expression.



Now learn about various types of operators.

Arithmetic Operators

These operators are used to do arithmetical operations.

Operator	Name	Description	Example (x=7 and y=3)	Output
+	Addition	Adds values on either side of the operator	$x + y$	10
-	Subtraction	Subtracts right hand operand from left hand operand	$x - y$	4
*	Multiplication	Multiplies values on either side of the operator	$x * y$	21
/	Division	Divides left hand operand by right hand operand	x / y	2.3333335
%	Modulus	Divides left hand operand by right hand operand and returns the remainder	$x \% y$	1
**	Exponentiation	Performs exponential (power) calculation on operands	$x ** y$	343
//	Floor or Integer division	Divides and returns the integer part from the result.	$x // y$	2

Relational Operators or Comparison Operators

These operators are used to compare two values to one another.

Operator	Name	Description	Example (x=8 and y=6)	Output
==	Equal	It checks if the values of two operands are equal or not. If the values are equal, then the condition becomes true.	x == y	False
!=	Not equal	It checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	x != y	True
>	Greater than	It checks if the value of left operand is greater than the value of the right operand. If yes, then the condition becomes true.	x > y	True
<	Less than	It checks if the value of the left operand is less than the value of the right operand. If yes, then the condition becomes true.	x < y	False
>=	Greater than or equal to	It checks if the value of the left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	x >= y	True
<=	Less than or equal to	It checks if the value of the left operand is less than or equal to the value of the right operand. If yes, then the condition becomes true.	x <= y	False

Logical Operators

These operators are used to make decision on two conditions or more than two condition. Logical Operators are used with boolean values and return boolean value as output.

Operator	Name	Description	Example (x=2)	Output
and	AND	It returns true if both operands are true.	(x < 5) and (x < 10)	True
or	OR	It returns true if one of the operands is true.	(x < 5) or (x < 2)	True
not	NOT	It reverses the result, and returns false, if the result is true or vice versa.	not [(x < 5) and (x < 10)]	False

Assignment Operators

These operators are used to assign value to a variable.

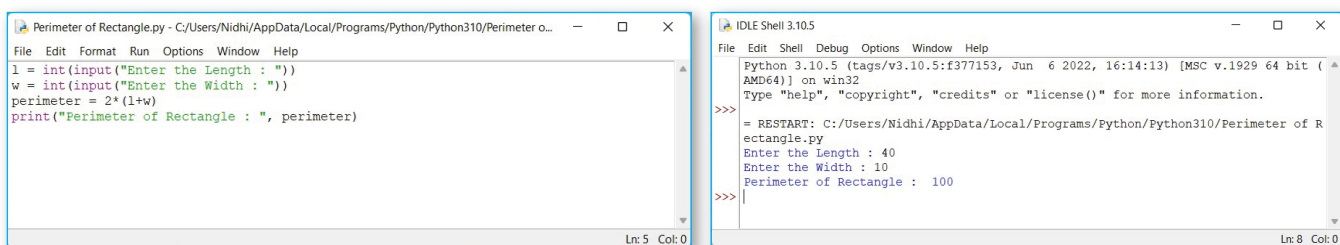
Operator	Name	Description	Example & Output (x=6)
=	Assignment	It assigns the value of the operand on the right side to the left side operand.	x = 6
+=	Addition assignment	It adds the right operand to the left operand and assigns the result to the left operand. x+=3 is equivalent to x=x+3.	x += 3 x=9
-=	Subtraction assignment	It subtracts the right operand from the left operand and assigns the result to the left operand. x-=3 is equivalent to x=x-3.	x -= 3 x=3
=	Multiplication assignment	It multiplies the right operand with the left operand and assigns the result to the left operand. x=3 is equivalent to x=x*3.	x *= 3 x=18
/=	Division assignment	It divides the left operand with the right operand and assigns the result to the left operand. x/=3 is equivalent to x=x/3.	x /= 3 2.0
%=	Remainder assignment	It takes the modulus of two operands and assigns the result to the left operand. x%=3 is equivalent to x=x%3.	x %= 3 x=0
//=	Floor division assignment	It performs floor division on operators and assigns the value to the left operand. x//=3 is equivalent to x=x//3.	x //= 3 x=2
**=	Exponentiation assignment	It performs exponential (power) calculations on operators and assigns the value to the left	x **= 3 x=216

Operator Precedence

An arithmetic expression without parentheses will be evaluated from left to right using the rule of precedence of operators.

Priority	Operator	Name
First	()	Parenthesis
Second	**	Exponent
Third	*, /, %, //	Multiplication, Division, Modulus, Floor Division
Fourth	+, -	Addition, Subtraction
Fifth	==, !=, >, <, >=, <=	Comparison
Sixth	=, +=, -=, *=, /=, %=, **=, //=	Assignment
Seventh	and, or, not	Logical

Write a program to use operators in Python.



```
File Edit Format Run Options Window Help
l = int(input("Enter the Length : "))
w = int(input("Enter the Width : "))
perimeter = 2*(l+w)
print("Perimeter of Rectangle : ", perimeter)
```

```
File Edit Shell Debug Options Window Help
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:/Users/Widhi/AppData/Local/Programs/Python/Python310/Perimeter of Rectangle.py
Enter the Length : 40
Enter the Width : 10
Perimeter of Rectangle : 100
>>>
```

Find on Google

Which type of applications can be developed in Python?

PUNCTUATORS

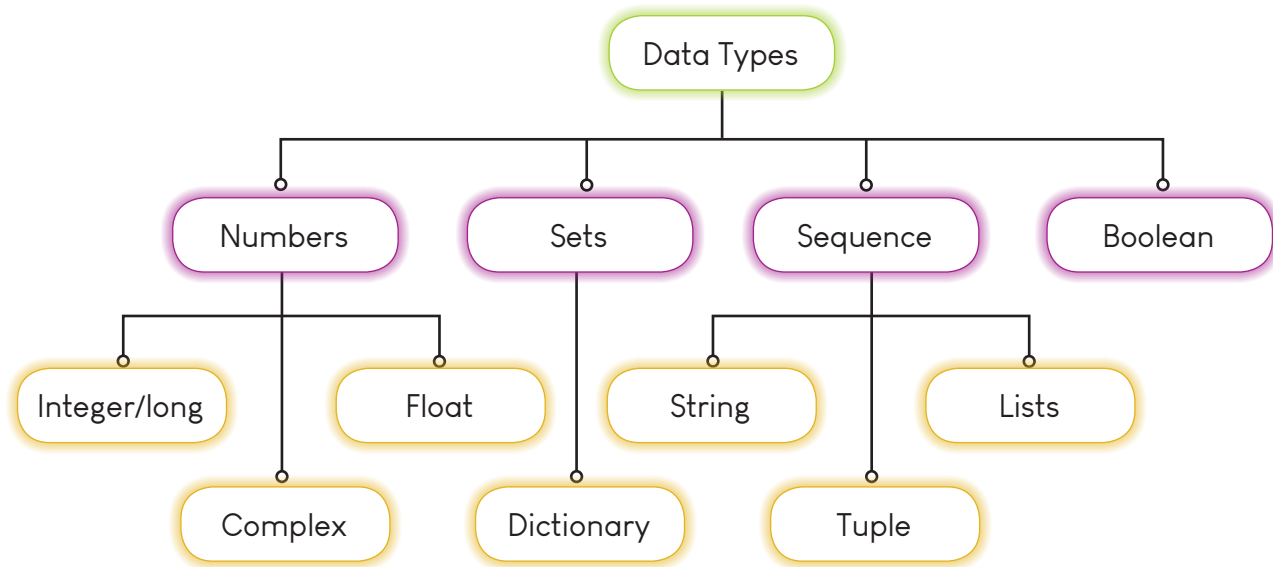
Punctuators are also called separators as they are used to separate lines of codes, variables, etc. The following characters are used as punctuators in Python: ` # \ () [] { } @ , : . ' =

- Brackets [] → indicate list.
- Parentheses () → indicate tuples, function calls, function parameters for grouping expressions etc.
- Comma (,) → is used as a separator in a function argument list or variable declaration.
- Colon (:) → indicates a labelled statement or conditional operator symbol.
- Equal sign (=) → is used to initialise the value of variable.



DATA TYPES

Data types are used to define the type of value a data can contain. Each variable in Python is associated with some data type. Each data type requires a different amount of memory and has some specific operations performed on it. Data types are divided into two categories as shown below:



NUMBER

Integer numbers, floating-point numbers and complex numbers are the built-in numeric data types of Python.

Integer/Long

Integers are whole numbers consisting of + or – signs. They do not have decimal places between them. For example, 87, -3 are called integer numbers.

Float

Float data type represents numbers that contain a decimal point. For example, 0.6, -3.257 are called floating-point numbers. These numbers can also be written using ‘e’ or ‘E’ to represent the power 10.

Complex

A complex is a number that is written in the form $a + bj$ or $a + bJ$. It is represented as follows: (real part) + (imaginary part) j.

SETS

Set is a mutable, unordered collection of values, of any type, with no duplicate element.

Dictionary

A dictionary is a collection of key-value pairs separated by commas (,) and enclosed within curly braces ({}). Each key is unique and is followed by a colon (:) that separates it from its corresponding value. For example: {"name": "Anairya", "age": 10, "grade": "4th"}.

SEQUENCE

A sequence is an ordered collection of items of similar or different data types. The three types of sequence data types are Strings, Lists and Tuples.

Strings

A string is a sequence of one or more characters put in single quotes, double quotes or triple quotes which is used to represent text-based information. The quotes are not a part of the string. For example, "hello", "Orange Education", 'Amit', '365'.

List

A list is a collection of data elements separated by comma (,) and enclosed within square brackets []. We can also create a list with the elements that are the same or have different data types. For example, [1,2,5], ['HELLO',45,87.6].

Tuple

A tuple is just like a list. It contains a group of elements that can be of different data types. The elements in the tuple are separated by commas (,) and enclosed in parentheses (). The difference between a list and a tuple is that list can be modified but tuples cannot be modified after it is created. For example, (7, 9.5, 3), ('a',4,3).

BOOLEAN

Boolean is a data type with two built-in values: **True** or **False**. They are used in logical evaluation. A True statement returns value 1 while a False statement returns value 0.



FIND DATA TYPE

Python allows us to find out the type of data used in a program by using type() function.

```
IDLE Shell 3.10.5
File Edit Shell Debug Options Window Help
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> #Program to find out type of data used in a program
>>> x = 7
>>> print("Data type of x is", type(x))
Data type of x is <class 'int'>
>>> y = 12.4
>>> print("Data type of y is", type(y))
Data type of y is <class 'float'>
>>> z = 3+5j
>>> print("Data type of z is", type(z))
Data type of z is <class 'complex'>
>>>
```




ERRORS IN PYTHON PROGRAMS

Errors are faults in a program. Errors prevent a program from executing accurately. There can be the following types of errors in a Python program:

SYNTAX ERRORS

A syntax error will occur when these rules and regulations are violated. For Example:

```
IDLE Shell 3.10.5
File Edit Format Run Options Window Help
>>> print'Hello'
SyntaxError: Missing parentheses in call to 'print'. Did you mean >>> print('Hello')?
>>> print('Hello')
Hello
```

LOGICAL ERRORS

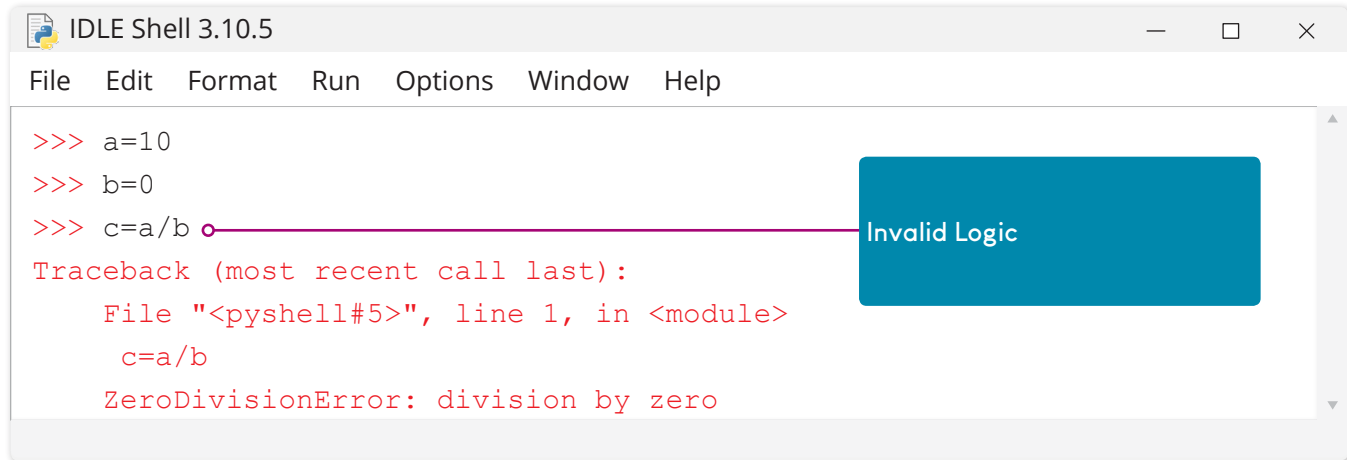
As the name suggests, these errors are related to the logic of the program. These errors are also known as **semantic errors**. They cause the program to behave incorrectly. They are the most difficult errors to fix but they do not usually crash the program. For example:

```
IDLE Shell 3.10.5
File Edit Format Run Options Window Help
>>> #Example of Logical Errors in a Program
>>> num1=float(input('Enter a number:'))
Enter a number:7
>>> num2=float(input('Enter another number:'))
Enter another number:8
>>> average=num1+num2/2
>>> print(average)
11.0
>>>
```

RUN-TIME ERRORS

Run-time errors in Python occur while the program is executing, causing it to crash or behave unexpectedly. These errors arise due to issues such as invalid operations, unavailable resources, or invalid inputs.

For example:



The screenshot shows the IDLE Shell 3.10.5 window. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code entered is:

```
>>> a=10
>>> b=0
>>> c=a/b
```

A red line points from the error message to the division operation in the code. The error message is:

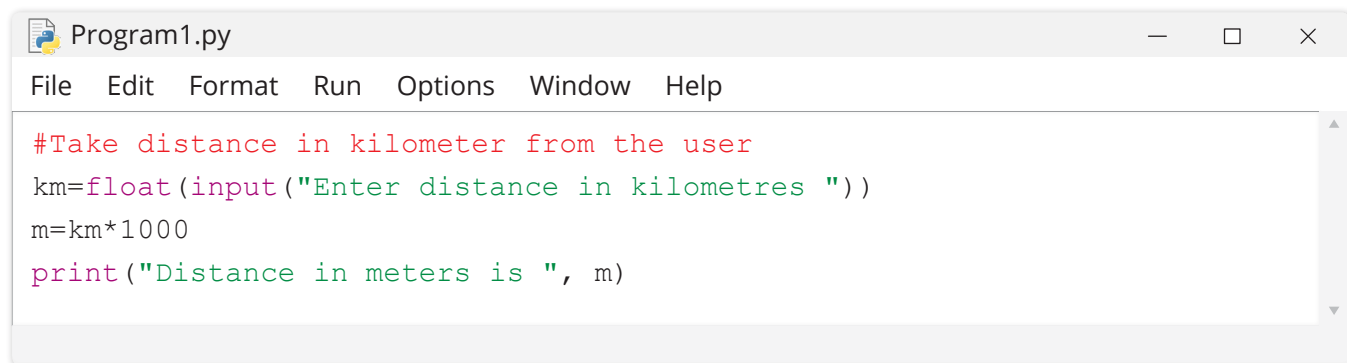
```
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    c=a/b
ZeroDivisionError: division by zero
```

A blue box with the text "Invalid Logic" is positioned to the right of the code.



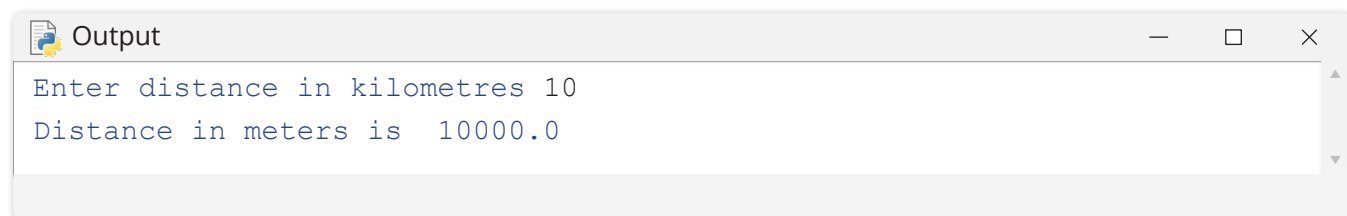
SOME MORE PROGRAMS

1. Write a program to take distance in Kilometres and convert it into Meters.



The screenshot shows the Program1.py code editor window. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code entered is:

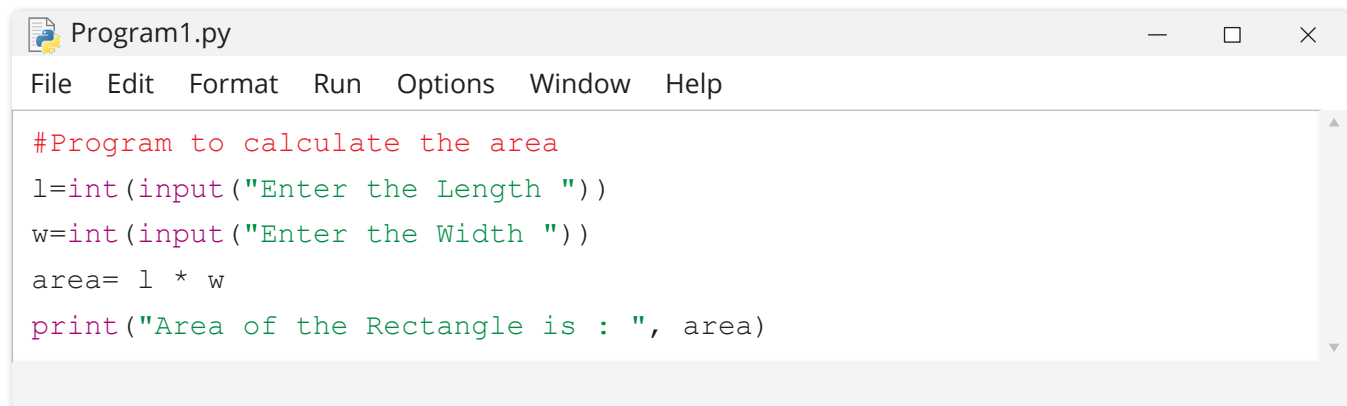
```
#Take distance in kilometer from the user
km=float(input("Enter distance in kilometres "))
m=km*1000
print("Distance in meters is ", m)
```



The screenshot shows the Output window. The text displayed is:

```
Enter distance in kilometres 10
Distance in meters is 10000.0
```

2. Write a program to take the length and width of a rectangle as input and calculate its area.



The screenshot shows the Program1.py code editor window. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code entered is:

```
#Program to calculate the area
l=int(input("Enter the Length "))
w=int(input("Enter the Width "))
area= l * w
print("Area of the Rectangle is : ", area)
```

```
Output
Enter the Length 10
Enter the Width 4
Area of the Rectangle is : 40
```