Introduction to Python

1. What is Python?
2. History of Python
3. Python Versions
4. Applications of Python
5. Python Software
6. Python Features

What is Python?

Python is a programming language.
Python is high level and object oriented programming language.
Python is multi paradigm programming language (POP,OOP, FOP,MOP).
Python is a general purpose programming language, A programming language used for developing different types of applications.
Python is interpreted programming language.
"python is a high level, object oriented, interpreted and general purpose multi paradigm programming language"

Python History

Python programming language is conceived in the year 1980.
The implementation of python is started in the year 1989.
The first version of python is released in the year 1991.
Python language is developed by a Dutch programmer "Guido Van Rossum".
Python language is developed at CWI (Centrum Wiskunde & Informatica).
Python is managed by a non profitable organization called PSF (Python Software Foundation).
Python programming language is written in C language.
Python language is derived from ABC Language.
This language is developed by inheriting features from various programming language.

1. C
2. C++
3. Modula-7
4. Perl

Python Versions

Python 1.x → 1.0,1.1,1.2,1.3,1.4,1.5,1.6
Python 2.x → 2.0,2.1,2.2,2.3,2.4,2.5,2.6,2.7
Python 3.x → 3.0,3.1,3.2,3.3,3.4,……. 3.12

There is no compatibility between major versions.

Python 3.12.4
3 → Major Version
12 → Minor Version
4→ Micro version

PEP → Python Enhancement proposal.
Python 1 & 2 Versions are out dated not used by software companies.
Companies are using Python 3.8, 3.9, 3.10,3.11,3.12

Python Applications

Python applications are nothing but, python used in real time for developing which type of software's or applications.

What is Library?

Library is pre-written program, with pre-written functionality.
Python libraries are in forms

1. Modules (Python Program)
2. Packages (Collection of modules)

Python language uses libraries for developing different type of applications.

1. Web Applications (Django, Flask, RestAPI)
2. Web Scraping (Beautiful SOAP,scrappy)
3. Data Science (Numpy, Pandas, Matplitlib, Keras, …)
4. AI (TensorFlow. SciPy. ...Seaborn. ...Scikit-learn. ..)
5. Automation/Testing (Selenium)
6. Windows Programming (Tkinter,wxpython,pyQT)
7. Scientific Application (SciPy, Scikit-learn)
8. CAD/CAM
9. Big Data or Data Eng (PySpark, Boto)
10. Cloud Computing (AWS) (Boto)
11. DevOPS (CI/CD)
12. Games Development (PyGame,..)
13. Mobile Development (Kivy)
14. Business Application
15. Education
16. Network Enabled Applications
17. Language Development
18. Cyber Security
19. Audio/Video Processing
20. Image Processing
21. Block Chain
22. Data Visualization

These libraries are available in one repository
www.pypi.org
PyPI stands Python Package Index.

Python Features

Python features are nothing but facilities provided by python to developers.

1. Easy or Simple
2. Free and Open Source
3. Large Standard Libraries
4. Platform Independent
5. High Level and Portable
6. Dynamic
7. Interpreted
8. Extensible
9. Embeddable
10. Object Oriented

Easy

1. **Less Coding**: python provides large number of libraries. Libraries provides predefined code. This code is used by programmer to perform a specific task.
2. **Easy Syntax**: In python statements are not terminated with ; and blocks are defined using curly braces
3. **Automatic Memory Management:** in python memory management is automatic, it is done by python. Unused memory is removed automatically by python (garbage collector).
4. **High Level :**All high level languages are in english

2. Free and Open Source

Python software is free to download

www.python.org

What is open source?
Source code of python is given public.

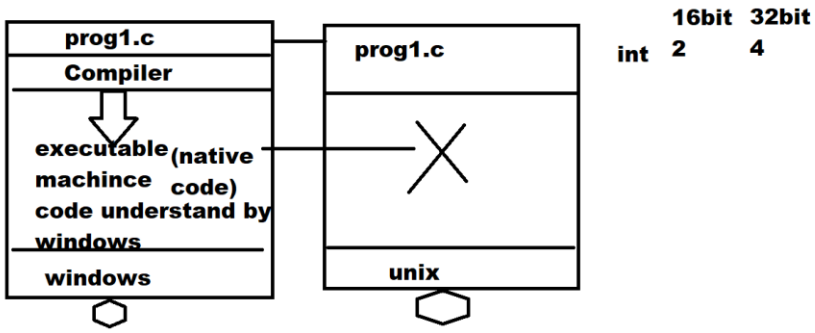
Advantage of open source is developing new languages, frameworks and integrating existing technologies.

3. **Large Standard Libraries**
Python provides large number of libraries and having huge community support.
www.pypi.org

4.**Platform Independent**

What is platform?
Platform is software which provide good environment for execution and development of applications or software's.
Operating System is called platform.

What is platform dependence?
In platform dependent programming languages development environment and execution environment must be same.



C,C++ are called platform dependent programming language.

1. Compiled code of C is platform dependent. Whenever C program is compiled, C compiler generates native code (binary code respective to OS).
2. Data representation in C language changes from one OS to another.

What is platform independence?
In platform independent programming languages development environment and execution environment may not be same.

What is byte code?

Compiled code of python source program is called byte code.

Byte code is virtual machine code, which understands by python virtual machine.

Byte code is not 0's and 1's. It is a collection of mnemonics (verbs).

Byte code is platform independent code.

What is PVM?

PVM stands for Python Virtual Machine.

PVM provides runtime environment for python programs.

PVM having a translator called interpreter, which translates byte code into executable machine code.

Python is an object oriented programming language, data is represented as objects and these objects are dynamic in size.

WORA/CORA

High Level and Portable

All high level languages are portable languages.

Portable languages allows develop and run applications or programs irrespective of hardware.

All high level languages are english like.

Dynamic

Programming languages are two types

1. Statically typed languages
2. Dynamically typed languages

Statically typed languages

C,C++,Java,.Net are called statically typed programming languages.

Statically typed programming languages variable declaration is required. Variables declared with specific data type.

Dynamically typed languages

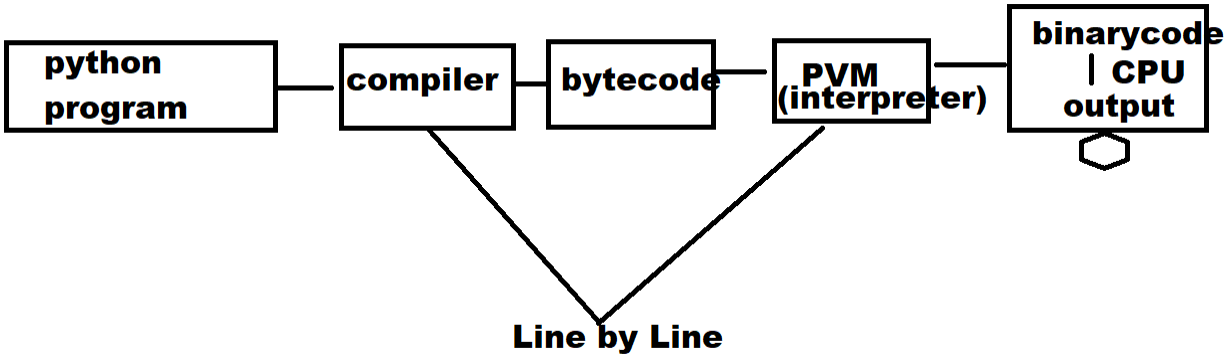
All scripting languages are dynamically typed programming languages.

In dynamically typed programming languages there is no variable declaration. The type of variable is based on value.

Interpreted Language

In python compiling and interpretation is done line by line.

Python program has to interpret every time before execution.

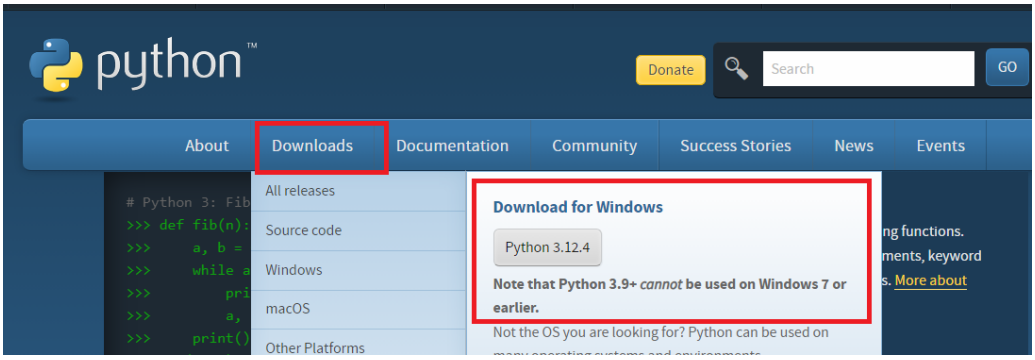


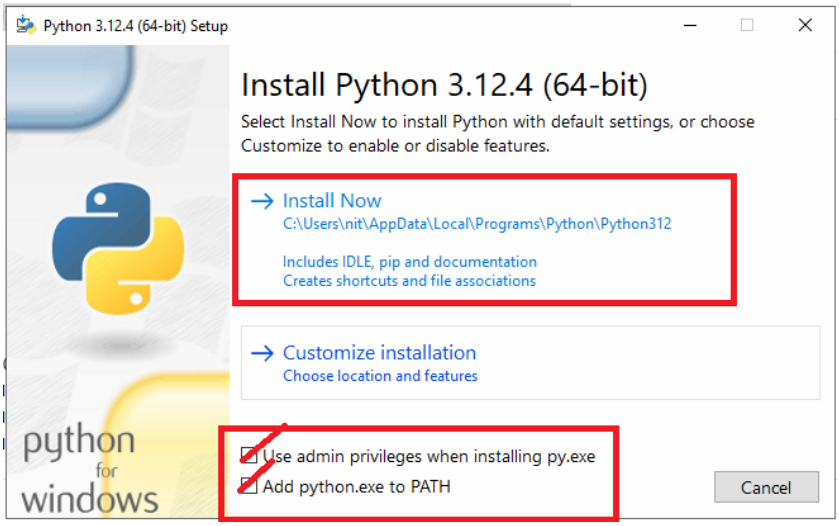
Extensible and Embeddable
Using python in other programming languages is called embeddable.
Using other languages code within python is called extensible.

Object oriented
Python is object oriented programming languages. In python programs can be written using OOP Concepts.

Python Software

Python software can be downloaded from www.python.org





Python software provides,

1. Python Compiler
2. PVM
3. Python Shell
4. IDLE → Integrated development Learning Environment (code editors/ide)
5. Standard Libraries
6. Tools (Debugger, Package installer)

Other IDE's

1. PyCharm
2. Jupiter
3. VSCode
4. Spyder
5. GoogleColab

Python implementations

1. Jython
2. IronPython
3. PyPY
4. CPython
5. MicroPython

Python Distributions

Python distribution is a python software bundle which consists of,

1. Python software
2. Editors
3. Application specific libraries

ActiveState ActivePython (commercial and community versions, including scientific computing modules)
pythonxy (Scientific-oriented Python Distribution based on Qt and Spyder)
winpython (WinPython is a portable scientific Python distribution for Windows)
Conceptive Python SDK (targets business, desktop and database applications)
Enthought Canopy
(a commercial distribution for scientific computing)
PyIMSL Studio (a commercial distribution for numerical analysis – free for non-commercial use)
Anaconda Python (a full Python distribution for data management, analysis and visualization of large data sets)
eGenix PyRun (a portable Python runtime, complete with stdlib, frozen into a single 3.5MB - 13MB executable file)

How to find python software installed or not?

1. Command prompt
   a. Search → CMD

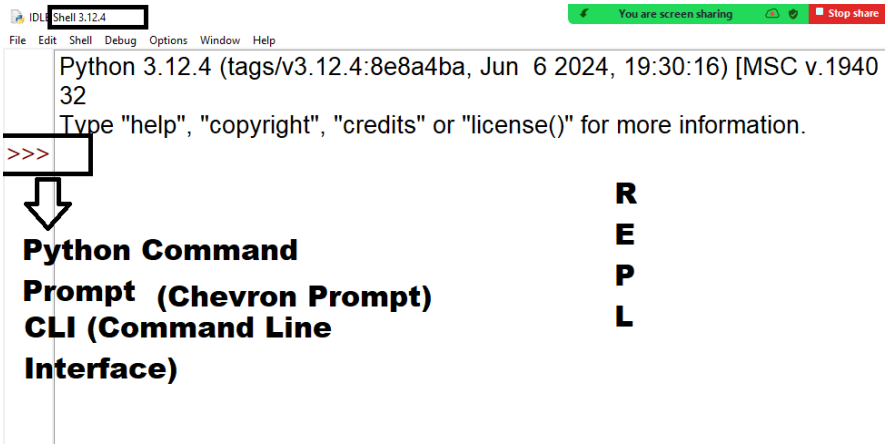
2. Search → IDLE

Python Working Modes

Python developers work with python in 2 different modes.
1. Interactive Mode
2. Scripting Mode/Programming Mode

Interactive Mode
In interactive mode, python developer work with python shell.

Python shell is command line interpreter (OR) Python shell is called REPL (Read-Evaluate-Print-Loop) tool.
Python shell is single line command line interface.
In interactive mode, python developer cannot write programs/scripts.



Scripting mode or programming mode
A program collection of instructions
A Program is python file, which contains set of instructions executed by computer. Every python program having extension .py

How to write a program in IDLE?

1. Open IDLE
2. File → New File
3. Write Program

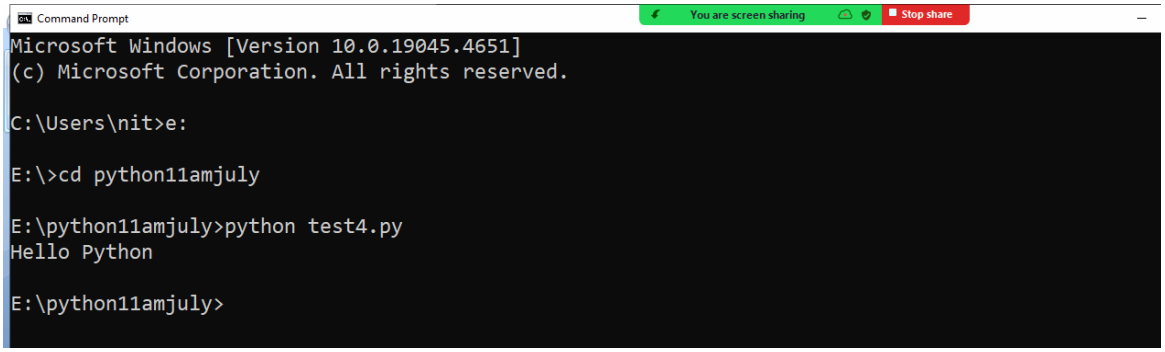
4. File → Save → test1.py
5. Run → Run Module

How to write and execute python program without using python editors?

1. Any Text Editor (Notepad)
2. Write program

3. Save the program with extension .py
4. Run the program

a. Open command prompt
b. Open location where program is saved



Python Language Fundamentals

1. Character set of Python
2. Tokens
    a. Keywords
    b. Identifiers
    c. Data Types
    d. Operators
    e. Literals
    f. Comments

Character set of python

Character set of python defines encoding and decoding standards.

1. ASCII
2. UNICODE

ASCII stands for American Standard Code for Information Interchange.

ASCII support 256 characters

1. Upper case (English)
2. Lower case (English)
3. Digits (0-9)
4. Special Characters

Upper case ASCII Values

A → 65

B → 66

…
Z → 90

Lower case ASCII Values
a → 97
b → 98
…
z → 122

Digits ASCII Values
0 → 48
1 → 49
2 → 50
…
9 → 57

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

UNICODE
UNICODE is a super set of ASCII
UNICODE support the characters in english and other languages (Hindi, Telugu,…)
UNICODE support more I M characters.

```
>>> a="ABC"
>>> x=12
>>> name="naresh"
>>> name
'naresh'
```

```
>>> name="నరేష్"
>>> name
'నరేష్'
>>> name="नरेश"
>>> name
'नरेश'
```

Tokens of python

What is Token?

Token is smallest individual unit within program.

These tokens

1. Keywords
2. Identifiers
3. Data Types
4. Literals
5. Operators
6. Comments

What is Token?

Token is smallest individual unit within program.

These tokens

7. Keywords
8. Identifiers
9. Data Types
10. Literals
11. Operators
12. Comments

Keywords OR Reserved words

Keywords are language related words. These words are used to perform a specific task or operation. These words are having special syntax and this syntax is understood by python translators.

How find keyword list in python?

```
>>> 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']
>>> len(keyword.kwlist)
35
```

```
>>> keyword.softkwlist
['_', 'case', 'match', 'type']
>>> len(keyword.softkwlist)
4
```

Keywords represent values

1. True
2. False
3. None

Keywords represent operators

1. and
2. or
3. not
4. is
5. in
6. del
7. as

Keywords represent control statements

1. if
2. else
3. elif
4. while
5. for
6. break
7. continue
8. return
9. pass

Keywords represent functions

1. def
2. lambda
3. nonlocal
4. global
5. yield

Keywords represent class

1. class
2. with

Keywords represent modules and packages

1. import

2. from

Keywords represent exception handling

1. try
2. except
3. finally
4. assert
5. raise

Keywords represent multithreading

1. async
2. await

Identifiers

Identifier is user defined word.
Identifier is used to identify programming elements.

1. Variable name
2. Function name
3. Class name
4. Constant name
5. Module name
6. Package name

Identifier is a word, which is created using alphabets (A-Z, a-z), digits (0-9) and special character (_)

Rules for defining identifiers

1. Identifier should not be keyword

```
>>> area=100
>>> area
100
>>> pass=50
SyntaxError: invalid syntax
>>> break=2
SyntaxError: invalid syntax
>>> rollno=10
>>> rollno
10
```

Note: python case-sensitive language, it finds the difference between uppercase and lowercase.

```
>>> A=100
>>> a=200
>>> A
100
>>> a
200
```

Identifiers

Identifier is user defined word.
Identifier is used to identify programming elements.

7. Variable name
8. Function name
9. Class name
10. Constant name
11. Module name
12. Package name

Identifier is a word, which is created using alphabets (A-Z, a-z), digits (0-9) and special character (_)

Rules for defining identifiers

2. **Identifier should not be keyword**

```
>>> area=100
>>> area
100
>>> pass=50
SyntaxError: invalid syntax
>>> break=2
SyntaxError: invalid syntax
>>> rollno=10
>>> rollno
10
```

Note: python case-sensitive language, it finds the difference between uppercase and lowercase.

```
>>> A=100
>>> a=200
>>> A
100
>>> a
200
```

3. **Identifier should not start with digit**

```
>>> a1=100
>>> a1
100
>>> a2=200
>>> a2
200
>>> 3a=300
SyntaxError: invalid decimal literal
```

4. **One special character is allowed within identifier _**

```
>>> _a=100
>>> _a
100
>>> __=200
>>> __
200
>>> _=500
>>> _
500
>>> amt$=5
SyntaxError: invalid syntax
>>> $amt=1
SyntaxError: invalid syntax
>>> a$b=10
SyntaxError: invalid syntax
>>> a_b=10
>>> a%b=1
SyntaxError: cannot assign to expression here. Maybe you meant
'==' instead of '='?
```

5. **There should not be any space between identifier name**

```
>>> rollno number=1
SyntaxError: invalid syntax
>>> student_rollno=1
>>> student_rollno
```

6. **The maximum length of identifier unlimited**

```
>>> aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa=10
>>> aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
10
```

7. **Identifier can be defined in uppercase or lowercase**

```
>>> a=100
>>> a
100
>>> A=300
>>> A
300
>>> a
100
>>> A
300
```

What is difference between keywords and identifiers?

Keyword	**Identifiers**
Keywords are predefined words	Identifiers are user defined words
Keywords are identified by python translator	Identifiers meaning understand by programmer
Keywords are used to perform specific task	Identifier is used to identify programming elements
Keywords are 35 +4 Soft keyword	Identifiers are used defined words, these can be unlimited

What is difference between keywords and softkeywords?

Keywords cannot be used as identifiers	Soft keywords can be used as identifiers

	_ match case type

Literals and Data types

Data types

Data types are used to reserve memory/space within main memory/RAM.
Data type represents for which type of data how much memory has to reserve.

Python standard data types are classified into 2 categories

1. Scalar Data types
2. Collection Data types

Scalar Data types are used to reserve memory for one value
Scalar data types are 5

1. int
2. float
3. complex
4. bool
5. NoneType

Collection data types are used to reserve memory for more than one value.

1. Sequences
    a. List
    b. Tuple
    c. Str
    d. Range
    e. Bytes
    f. bytearray
2. Sets
    a. Set
    b. frozenset
3. Mappings
    a. Dictionary

Python support 14 standards data types.

Literals

Literals are values or constants which never changed.
Python literals are divided into different types

1. Integer literals
2. Float literals
3. Boolean literals
4. Complex literal
5. String literal

Integer literal/value

Integer value is a numeric value.
Integer values can be whole numbers, even numbers, odd numbers, natural numbers, prime numbers,…
An integer value is numeric value which does not decimal part or precisions.
In python integer numbers are represented in memory using **int data type.**

What is variable?
Variable is an identifier used to identify values
Variable is a named memory location which contains data.
Every variable is bind with data type.
The value of variable is not fixed it changes.

Example:

```
>>> a=4
>>> a
4
>>> type(a)
<class 'int'>
>>> a=10
>>> a
10
>>> type(a)
<class 'int'>
>>> a=1.5
>>> type(a)
<class 'float'>
```

Size of int data type is unlimited. PVM reserve the space based on value size.

```
>>> b=9
>>> import sys
>>> sys.getsizeof(b)
28
>>> c=999999999999
>>> sys.getsizeof(c)
32
>>>
d=99999999999999999999999999999999999999999999999999999999
>>> sys.getsizeof(d)
52
```

In python integer values/literals represented in 4 formats.

1. Decimal Integer
2. Octal Integer
3. Hexadecimal Integer
4. Binary Integer

Decimal Integer

An integer value with base 10 is called decimal integer.
This decimal integer is created using digits range from 0-9
This decimal integer is prefix with + or –
Decimal integer should not prefix with 0
By default integers are in decimal format.

(25)
10

$5x10^0 + 2x10^1$

5+20 = 25

```
>>> a=0123
SyntaxError: leading zeros in decimal integer literals are not
permitted; use an 0o prefix for octal integers
>>> b=+786
>>> c=-987
>>> type(b)
<class 'int'>
```

```
>>> type(c)
<class 'int'>
>>> b
786
>>> c
-987
>>> e=1,250
>>> type(e)
<class 'tuple'>
>>> e
(1, 250)
```

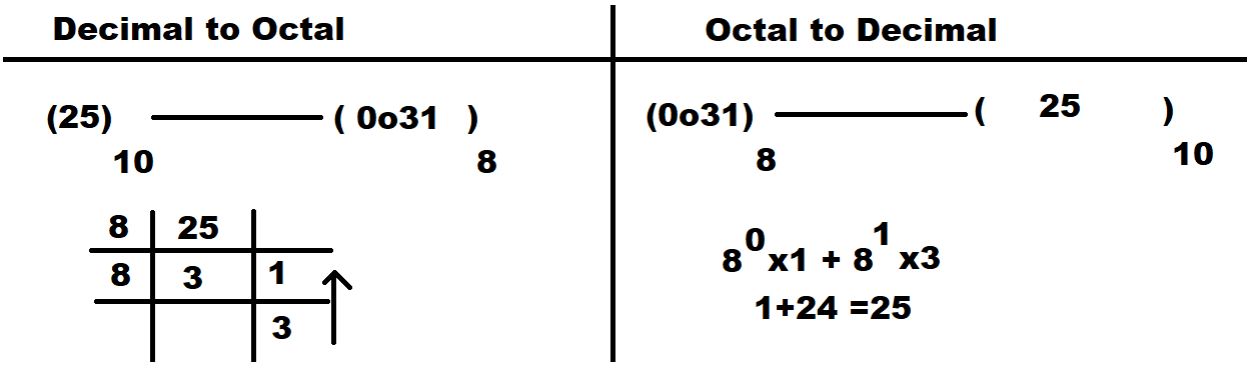
Numeric value allows one special character for grouping digits _ (underscore). This character is allowed in between.

```
>>> e=1,250
>>> type(e)
<class 'tuple'>
>>> e
(1, 250)
>>> f=1_250
>>> f
1250
>>> g=1_50_000
>>> g
150000
>>> h=250_
SyntaxError: invalid decimal literal
>>> i=_250
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    i=_250
NameError: name '_250' is not defined
```

type()
type is a predefined, which returns type of variable (OR) type of object/value hold by variable

Octal Integer
An integer value with base 8 is called octal integer.

This integer is prefix with 0o or 0O.
Octal integer is represented using digits range from 0-7



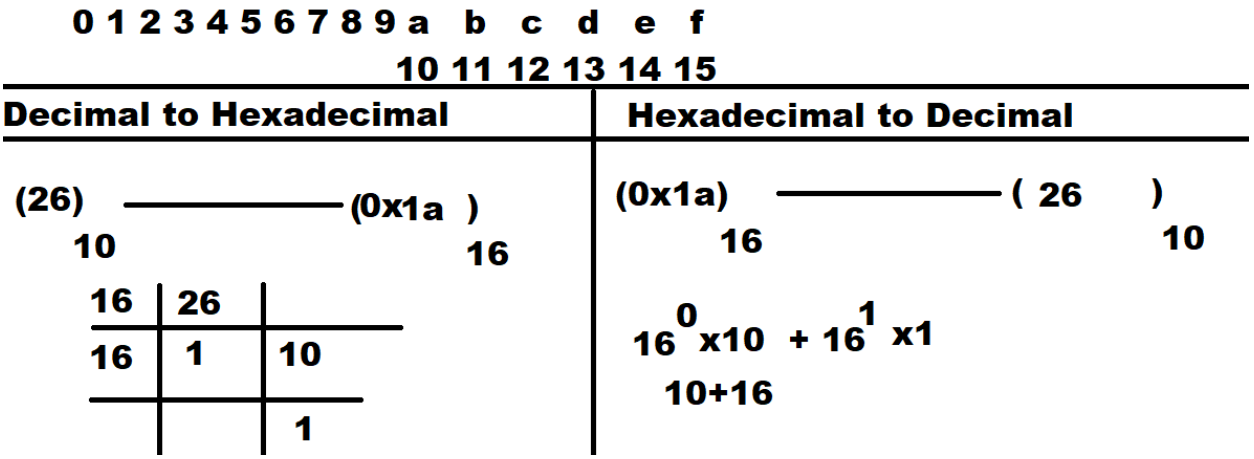

```
>>> a=0o31
>>> a
25
>>> b=0o78
SyntaxError: invalid digit '8' in octal literal
>>> c=0o999999
SyntaxError: invalid digit '9' in octal literal
>>> d=0O1208
SyntaxError: invalid digit '8' in octal literal
>>> type(a)
<class 'int'>
```

Hexadecimal integer

An integer value with base 16 is called hexadecimal integer.
This integer is created using digits range from 0-9, a-f/A-F
This integer is prefix with 0x or 0X.

Applications of hexadecimal integer

1. Color values
2. Memory Addresses
3. Register Address




```
>>> a=0x1a
>>> a
26
>>> b=0xa
>>> b
10
>>> c=0Xf
>>> c
15
>>> d=0xff
>>> d
255
>>> e=0xbad
>>> e
2989
>>> h=0xjava
SyntaxError: invalid hexadecimal literal
```

Binary Integer

An integer value with base 2 is called binary integer.
This integer is created using digits 0 and 1
This integer is prefix with 0b or 0B.

Applications of binary integer

1. Internal representation of integer is binary
2. Images Processing
3. Audio/Video Processing
4. Embedded Applications (Logic Gates)

Decimal to Binary

$(12)_{10}$ ——— $(0b\ 1100)_2$

2	12	
2	6	0
2	3	0
2	1	1
		1

Binary to Decimal

$(0b1100)_2$ ——— $(12)_{10}$

$2^0 x\ 0 + 2^1 x0 + 2^2 x1 + 2^3 x1$

0+0+4+8

```
>>> a=0b1100
>>> a
12
>>> b=0B1010
>>> b
10
>>> c=0b101
>>> c
5
>>> d=0b102
SyntaxError: invalid digit '2' in binary literal
```

Python provides base conversion functions

1. Oct() → return octal value of given integer
2. Hex() → return hexadecimal value of given integer
3. Bin() → return binary value of given integer

```
>>> oct(8)
'0o10'
>>> oct(0xa)
'0o12'
>>> oct(0b1100)
'0o14'
>>> hex(10)
'0xa'
>>> hex(14)
'0xe'
>>> hex(15)
'0xf'
>>> hex(26)
'0x1a'
```

```
>>> hex(0b1010)
'0xa'
>>> hex(0o12)
'0xa'
>>> bin(10)
'0b1010'
>>> bin(0xa)
'0b1010'
>>> bin(0o12)
'0b1010'
```

Float literals

Float value is numeric value with decimal part or fractional part.
In python float value is represented in 2 formats or notations

1. Fixed notation
2. Exponent notation

Float value in python is represented using "float" data type.
Default representation of float value is in fixed notation.

```
>>> a=1.5
>>> type(a)
<class 'float'>
>>> b=0.5
>>> type(b)
<class 'float'>
>>> c=0.0
>>> type(c)
<class 'float'>
>>> d=0
>>> type(d)
<class 'int'>
```

The size of float data type is not unlimited, it is fixed. The size of float is 8bytes.

How to find information about float data type?

```
>>> import sys
>>> sys.float_info
sys.float_info(max=1.7976931348623157e+308, max_exp=1024,
max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021,
```

```
min_10_exp=-307, dig=15, mant_dig=53,
epsilon=2.220446049250313e-16, radix=2, rounds=1)
>>>
```

In exponent notation float value uses on special character "e"/"E", value of e is 10

```
>>> x=15e-1
>>> x
1.5
>>> y=1.5e2
>>> y
150.0
>>> z=1.5e-2
>>> z
0.015
n1=1.7976931348623157e+308
>>> n1
1.7976931348623157e+308
>>> n2=1.7976931348623157e+309
>>> n2
inf
>>> n2=1.1234567891234567891234567 89
>>> n2
1.1234567891234568
>>> n3=1.1231231231231231231231231231231 23
>>> n3
1.1231231231231231
```

Complex literals

Complex literal is numeric value which represents complex number.

Complex number in python is represented using complex data type.

Syntax: a+bj

```
>>> c1=1+2i
SyntaxError: invalid decimal literal
>>> c1=1+2j
```

```
>>> c2=3j
>>> c3=5
type(c1),type(c2),type(c3)
(<class 'complex'>, <class 'complex'>, <class 'int'>)
>>> c1.real
1.0
>>> c1.imag
2.0
>>> c1.conjugate
<built-in method conjugate of complex object at
0x000002E04D4175B0>
>>> c1.conjugate()
(1-2j)
>>> c1
(1+2j)
```

Boolean Literals

In python Boolean values/literals are represented using bool data type. These Boolean values are represented using two keywords.

1. True
2. False

```
>>> b1=True
>>> type(b1)
<class 'bool'>
>>> b2=False
>>> type(b2)
<class 'bool'>
>>> b1
True
>>> b2
False
10+20
30
1.5+2.5
4.0
>>> True+True
2
>>> True+False
1
```

```
>>> 10+True
11
>>> 10-True
9
>>> 10+False
10
```

None Literal
None represents no value
None is a keyword in python
None value is represented using NoneType

```
>>> name=None
>>> rollno=1
>>> rollno
1
>>> name
>>> type(rollno),type(name)
(<class 'int'>, <class 'NoneType'>)
```

String
String is a collection of characters.
String is sequence data type.
String is non numeric data type.
“str” data type is used to represent string value

String value is represented in,

1. Single quotes
2. Double quotes
3. Triple single quotes or double quotes

String is a collection of characters, these characters can be

1. Alphabets
2. Digits
3. Special character

```
>>> name='rama rao'
>>> print(name)
rama rao
>>> fname="raja rao"
```

```
>>> print(fname)
raja rao
>>> course='''python'''
>>> print(course)
python
>>> remarks='"""Module-1and Module2"""'
>>> print(remarks)
Module-1and Module2
type(name),type(fname),type(course),type(remarks)
(<class 'str'>, <class 'str'>, <class 'str'>, <class 'str'>)
>>> a=12
>>> b="12"
type(a),type(b)
(<class 'int'>, <class 'str'>)
>>> c=1.5
>>> d="1.5"
>>> type(c),type(d)
(<class 'float'>, <class 'str'>)
>>> n1="10"
>>> n2="15"
>>> n1*n2
Traceback (most recent call last):
  File "<pyshell#91>", line 1, in <module>
    n1*n2
TypeError: can't multiply sequence by non-int of type 'str'
>>> n1-n2
Traceback (most recent call last):
  File "<pyshell#92>", line 1, in <module>
    n1-n2
TypeError: unsupported operand type(s) for -: 'str' and 'str'
>>> 10-5
5
>>> "10"-"5"
Traceback (most recent call last):
  File "<pyshell#94>", line 1, in <module>
    "10"-"5"
TypeError: unsupported operand type(s) for -: 'str' and 'str'
>>> "10"+"5"
'105'
>>> "11"+"22"
```

```
'1122'
>>> 11+22
33
```

Within single quotes we can represent single line string. Single quotes we can embed/insert double quotes
Within double quotes we can represent single line string. Double quotes we can embed/insert single quotes.
Triple double quotes or single quotes are used to represent multiline string.

Within single quotes we can represent single line string. Single quotes we can embed/insert double quotes
Within double quotes we can represent single line string. Double quotes we can embed/insert single quotes.
Triple double quotes or single quotes are used to represent multiline string.

```
>>> s1='python is a "easy" language'
>>> print(s1)
python is a "easy" language
>>> s2="python is a 'dynamic' language"
>>> print(s2)
python is a 'dynamic' language
>>> s3='python is 'compiled' language'
SyntaxError: invalid syntax
>>> s4="python is "compiled" language"
SyntaxError: invalid syntax
>>> s5='python
SyntaxError: unterminated string literal (detected at line 1)
>>> s6="python
SyntaxError: unterminated string literal (detected at line 1)
>>> s7='''python
is
programming
... language'''
>>> print(s7)
python
is
programming
```

```
language
>>> s8="""python
... is
... a
... programming
... language"""
>>> print(s8)
python
is
a
programming
language
```

Escape Sequences or characters

\n	New line
\\	\Backslash
\'	‘
\"	“
\t	Tab space
\b	Backspace
\v	Vertical tab space

```
>>> s1='this is \'python\' language'
>>> print(s1)
this is 'python' language
>>> s2="python is a \"programming\" language"
>>> print(s2)
python is a "programming" language
>>> s3='python\njava\noracle'
>>> print(s3)
python
java
oracle
>>> s4="python\nprogramming\nlanguage"
>>> print(s4)
python
programming
language
>>> s5="rollno\tname\tcourse"
>>> print(s5)
```

```
rollno	name	course
>>> s6="\\"
>>> print(s6)
\
```

Every program required 3 things

1. Input
2. Process
3. Output

Input is the information given to program. This input is given to the program or application using various input devices or applications.

This input data/information is processed by program.
Processing is nothing performing some operations on input data.

Output is nothing but information given by program.

print()

print() is in-built function in python (OR) library function.
print is a standard output function, it used to print/output data or information on console/monitor/file.

Note: by default every python program import a module or library called built-ins

How to find content of built-ins?

```
>>> dir(__builtins__)
```

Example:

```
import math
print("Hello Python")
print(10)
print(10+20)
print(math.factorial(5))
```

Output

```
Hello Python
```

10
30
120
print() receives 5 inputs

1. args
2. sep
3. end
4. file
5. flush

print() receives 5 inputs

1. args
2. sep
3. end
4. file
5. flush

This function is used to print one or more than one value. These values are given to print function by separating with , but print function print these values using space as a separator.

Example:

```
print(10)
print(10,20)
print(10,20,30,40,50)
print(1,1.5,"python",1+2j,True,None)
print()
print("Hello")
print()
print("Bye")
```

Output

```
10
10 20
10 20 30 40 50
1 1.5 python (1+2j) True None

Hello

Bye
```

Sep

Print function uses separator when it prints more than one value. default separator used by print function is space. This separator can be input while executing print function.

Example

```
print(10,20,sep=',')
print(100,200,300)
print(100,200,300,sep='$')
print(100,200,300,400,sep="nit")
print(100,200,300,400,sep="\t")
print(100,200,300,400,sep="\n")
print(10,sep=":")
```

Output

```
10,20
100 200 300
100$200$300
100nit200nit300nit400
100    200    300    400
100
200
300
400
10
```

end

the default value of end is "\n" (newline). This end value is inserted at the end of printing.

Example

```
print(100)
print(200)
print(300,end=':')
print(400)
print(500,600,end=".")
print(700)
print(100,end='&')
```

```
print(200)
```

Output

```
100
200
300:400
500 600.700
100&200
```

Example:

```
print(10,20,30,40,sep=",",end='$')
print(100,200,300,400,end='.')
print(500)
print(600)
```

Output

```
10,20,30,40$100 200 300 400.500
600
```

Example

```
rollno=1
name="naresh"
course="python"
fee=5000.0

print(rollno)
print(name)
print(course)
print(fee)

print(rollno,name,course,fee,sep="\n")
print("rollno",rollno)
print("student name",name)
print("course ",course)
print("fee ",fee)
```

Output

```
1
naresh
python
5000.0
```

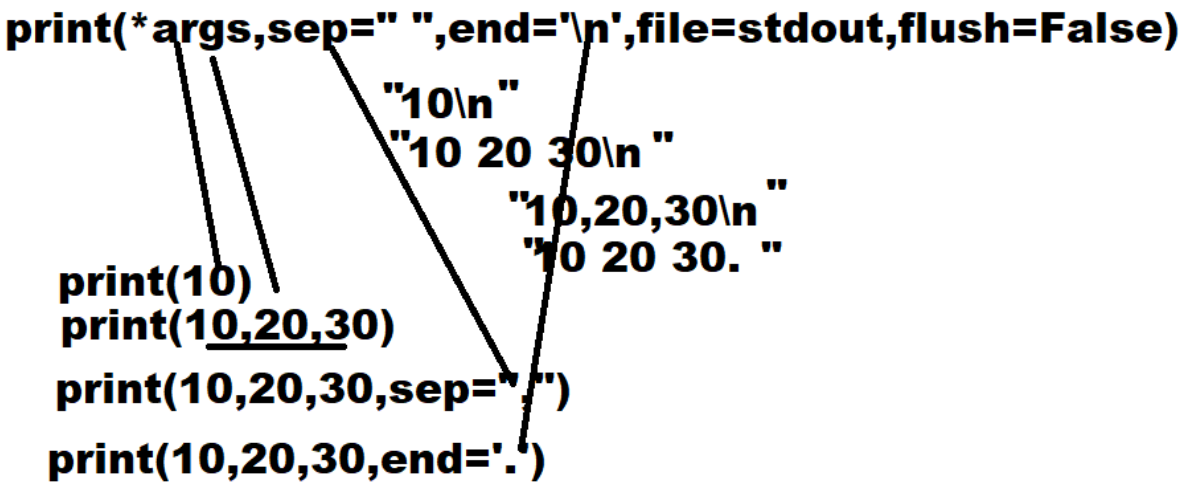
```
1
naresh
python
5000.0
rollno 1
student name naresh
course  python
fee  5000.0
```

Example

```
f=open("file1","w")
print(100,200,file=f)
print(1000,2000,file=f,flush=True)
```

Output

Output is saved inside file1



Comments

Comments are ignored by python translator.
In python comments are defined using #
It is used to marks single line as comment.

```
# this is my first program

#hold account number of customer
accno=1
#hold customer name
```

```
cname="naresh"

balance=50000 # hold customer balance

print(accno,cname,balance)
```

input()
input() is a predefined function in python.
A program reads data during runtime using input() function.

Syntax: input([prompt])
Prompt is a message displayed before input value.

Example:

```
a=input("Enter value of a ")
b=input("Enter value of b ")
print(a,b)
c=input()

rollno=input("Rollno :")
name=input("Name :")
course=input("Course :")
print(rollno,name,course)
```

Output

```
Enter value of a 10
Enter value of b 20
10 20
30
Rollno :101
Name :naresh
Course :python
101 naresh python
```

A program read a value from end user with help of input() function.
Using input end user can input only one value
Input() function allows to input value of type string (OR) input function read value of type string.

Example: uname=input("UserName :") pwd=input("Password :") print(uname) print(pwd)	**Example:** a=input("enter value of a") b=input("enter value of b") print(a,b)
Example a=input("enter value of a") print(a,type(a)) b=input("enter value of b") print(b,type(b))	**Example** a='10' b='1.5' c='1+2j' d='True' e='python' print(a,b,c,d,e) print(type(a),type(b),type(c),type(d),type(e))

Type casting or Type Conversion

It is a process of converting one type of value to another type. This conversion is done using type conversion functions provided by python.

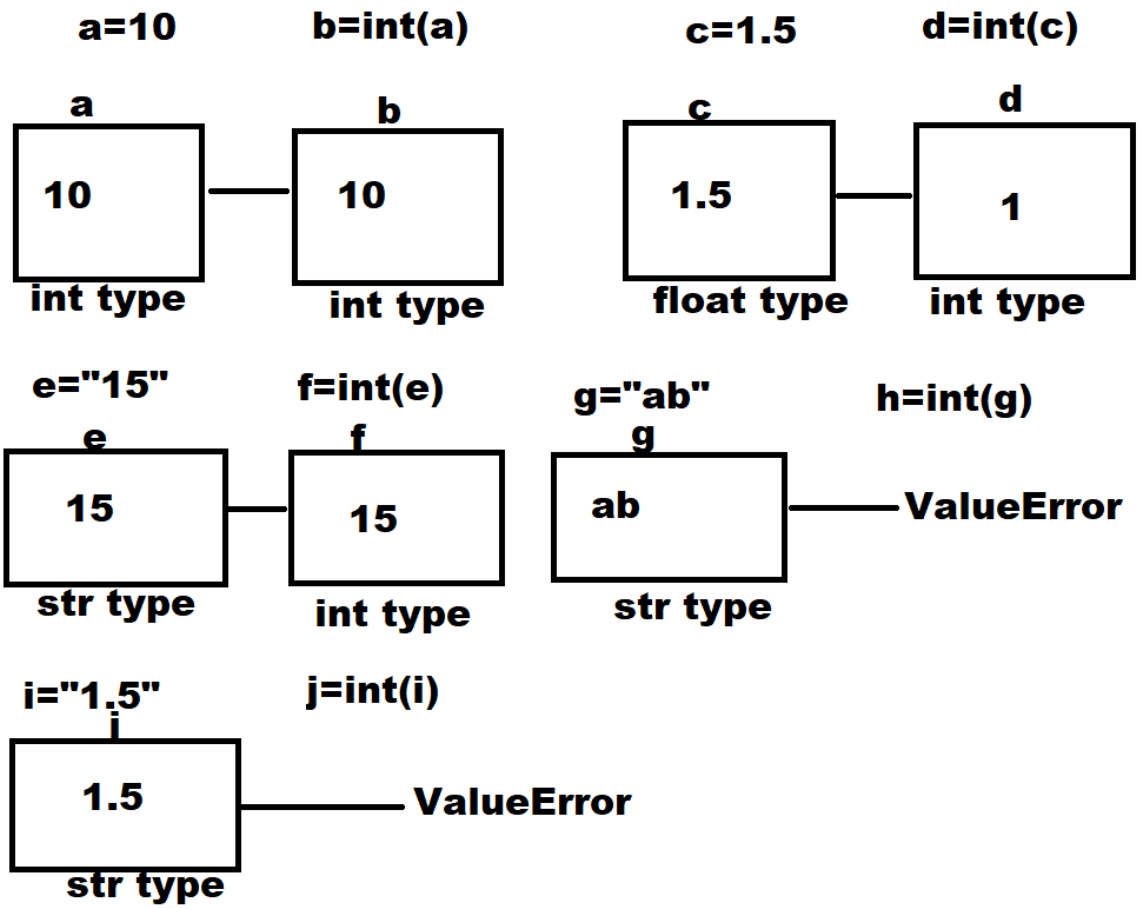
1. int()
2. float()
3. complex()
4. bool()
5. str()

int()

This function performs the following conversions

1. int to int
2. float to int
3. string to int
4. bool to int

syntax: int(value)



Example:

```
a=10
b=int(a)
print(a,b,type(a),type(b))

c=1.5
d=int(c)
print(c,d,type(c),type(d))

e="15"
f=int(e)
print(e,f,type(e),type(f))

#g="ab"
#h=int(g)

#i="1.5"
#j=int(i)
```

```
k="0xab"
l=int(k,base=16)
print(k,l,type(k),type(l))

m="0b101"
n=int(m,base=2)
print(m,n,type(m),type(n))

o="0o12"
p=int(o,base=8)
print(o,p,type(o),type(p))

q=True
r=int(q)
print(q,r,type(q),type(r))

s=False
t=int(s)
print(s,t,type(s),type(t))


#u=1+2j
#v=int(u)
```

Output

```
10 10 <class 'int'> <class 'int'>
1.5 1 <class 'float'> <class 'int'>
15 15 <class 'str'> <class 'int'>
0xab 171 <class 'str'> <class 'int'>
0b101 5 <class 'str'> <class 'int'>
0o12 10 <class 'str'> <class 'int'>
True 1 <class 'bool'> <class 'int'>
False 0 <class 'bool'> <class 'int'>
```

Example:

```
# Write a program to input name,subject1,subject2 marks
# and calculate total marks

name=input("Enter Student Name :")
sub1=int(input("Enter Subject1 Marks :"))
```

```
sub2=int(input("Enter Subject2 Mraks :"))

# 1st Method
#total=int(sub1)+int(sub2)

#2nd method
total=sub1+sub2

print("StudentName ",name)
print("Subject1 Marks ",sub1)
print("Subject2 Marks ",sub2)
print("Total Marks ",total)
```

Output

```
Enter Student Name :naresh
Enter Subject1 Marks :89
Enter Subject2 Mraks :67
StudentName  naresh
Subject1 Marks  89
Subject2 Marks  67
Total Marks  156
```

Example

```
# Write a program to swap two integer numbers

a=int(input("Enter value of a "))
b=int(input("Enter value of b "))


print('Before Swaping',a,b)
c=a
a=b
b=c
print('After Swaping ',a,b)
a,b=b,a
print('After Swaping ',a,b)
a=a+b
b=a-b
a=a-b
print('After Swaping ',a,b)
```

Output

```
Enter value of a 10
Enter value of b 20
Before Swaping 10 20
After Swaping  20 10
After Swaping  10 20
After Swaping  20 10
```

float() function

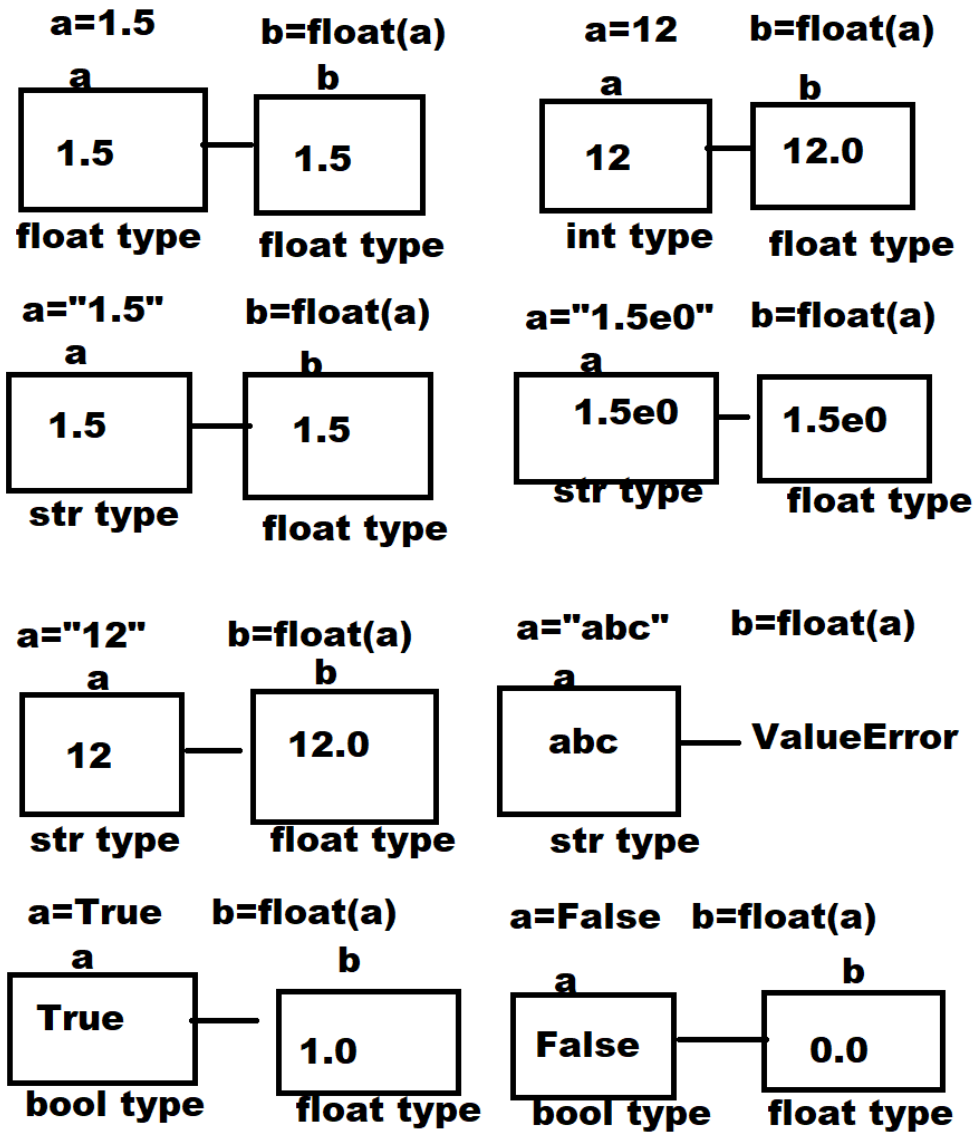
This function performs the following conversions

1. float to float
2. int to float
3. string to float
4. bool to float

float() function

This function performs the following conversions

5. float to float
6. int to float
7. string to float
8. bool to float



Example:

```
a=1.5
b=float(a)
print(a,b,type(a),type(b))

a=10
b=float(a)
print(a,b,type(a),type(b))

a="1.5"
b=float(a)
```

```
print(a,b,type(a),type(b))

a="15e0"
b=float(a)
print(a,b,type(a),type(b))

a="12"
b=float(a)
print(a,b,type(a),type(b))

a=True
b=float(a)
print(a,b,type(a),type(b))

#a=1+2j
#b=float(a)

#a="abc"
#b=float(a)
#print(a,b,type(a),type(b))
```

Output

```
1.5 1.5 <class 'float'> <class 'float'>
10 10.0 <class 'int'> <class 'float'>
1.5 1.5 <class 'str'> <class 'float'>
15e0 15.0 <class 'str'> <class 'float'>
12 12.0 <class 'str'> <class 'float'>
True 1.0 <class 'bool'> <class 'float'>
```

Example

```
# Write a program to find area of triangle
# area=0.5*base*height

# Input
base=float(input("Enter Base of the Triangle :"))
height=float(input("Enter Height of the Triangle :"))

# Process
area=0.5*base*height
```

```
#Output
print("Area of triangle is ",area)
```

Output

```
Enter Base of the Triangle :1.2
Enter Height of the Triangle :1.4
Area of triangle is  0.84
```

Example

```
# Write a program to add two float values

a=input("Enter First Float Value :")
b=input("Enter Second Float Value :")

print(a,b,type(a),type(b))

c=float(a)+float(b)
print("sum of ",a,b,"is",c)

name="Naresh"
print("my","name","is",name)
```

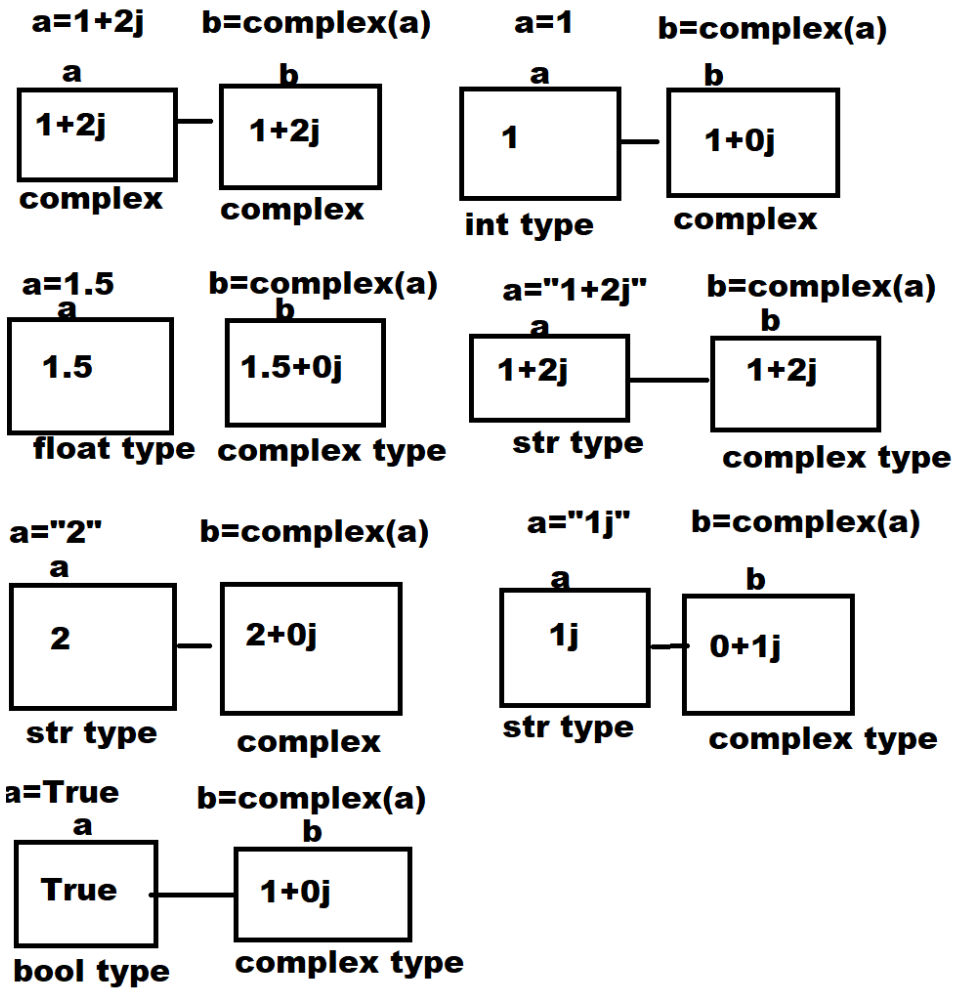
Output

```
Enter First Float Value :1.5
Enter Second Float Value :1.2
1.5 1.2 <class 'str'> <class 'str'>
sum of  1.5 1.2 is 2.7
my name is Naresh
```

complex() function

This function performs the following conversions

1. complex to complex
2. int to complex
3. float complex
4. string to complex
5. bool to complex



Example:

```
# Write a program to add two complex numbers

comp1=complex(input("Enter First Complex Number :"))
comp2=complex(input("Enter Second Complex Number :"))

comp3=comp1+comp2
print(comp1,comp2,comp3)
```

Output

```
Enter First Complex Number :1+2j
Enter Second Complex Number :1+3j
(1+2j) (1+3j) (2+5j)
```

str() function

This function perform the following conversion

1. string to string
2. int to string
3. float to string
4. complex to string
5. bool to string

Syntax: str(input)

Example:

```
# int to string
a=10
b=str(a)
print(a,b,type(a),type(b))

#float to string
a=1.5
b=str(a)
print(a,b,type(a),type(b))

#complex to string
a=1+2j
b=str(a)
print(a,b,type(a),type(b))

#string to string
a="PYTHON"
b=str(a)
print(a,b,type(a),type(b))

#bool to string
a=True
b=str(a)
print(a,b,type(a),type(b))
```

Output

```
10 10 <class 'int'> <class 'str'>
1.5 1.5 <class 'float'> <class 'str'>
(1+2j) (1+2j) <class 'complex'> <class 'str'>
PYTHON PYTHON <class 'str'> <class 'str'>
True True <class 'bool'> <class 'str'>
```

bool() function

Boolean function perform the following conversions

1. int to bool
2. float to bool
3. string to bool
4. complex to bool

Example:

```
>>> a=bool(1)
>>> print(a)
True
>>> b=bool(120)
>>> print(b)
True
>>> c=bool(-1)
print(c)
True
>>> d=bool("A")
>>> print(d)
True
>>> e=bool("0")
>>> print(e)
True
>>> f=bool(0)
>>> print(f)
False
>>> g=bool(0.1)
>>> print(g)
True
>>> h=bool(0.0)
>>> print(h)
False
>>> i=bool(1+2j)
>>> print(i)
True
>>> j=bool(0+1j)
>>> print(j)
True
```

```
>>> k=bool(0+0j)
>>> print(k)
False
>>> x=bool("False")
>>> print(x)
True
>>> y=bool("true")
>>> print(y)
True
>>> bool(None)
False
>>> bool("")
False
>>> bool('      ')
True
```

Operators

What is operator?

Operator is a special symbol having specific meaning in programming language.
Operator is a special symbol used to perform operations on data.

Based on the operands on which it performs operation, the operators are classified into 3 categories.

1. Unary Operators
2. Binary Operators
3. Ternary Operators

Unary operator required one operand for evaluation
Binary operator required two operands for evaluation
Ternary operator required three operands for evaluation

Types of Operators

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Membership Operators
5. Identity Operators
6. Compounds Assignment Operators

7. Bitwise Operators
8. Conditional Operators
9. Walrus Operators

Arithmetic Operators
These operators are used to perform arithmetic operations. All arithmetic operators are binary operators

Operator	Description
+	Addition or Concatenation
-	Subtraction
*	Multiplication or repeating
/	Float division
//	Floor division
%	Modulo or Modular
**	Power of operator

+ operator
This operator is used to perform two operations
1. Adding numbers
2. Concatenation of sequences (collection)

Example:
```
n1=10
n2=20
n3=n1+n2
print(n1,n2,n3)
f1=1.5
f2=1.2
f3=f1+f2
print(f1,f2,f3)
c1=1+2j
c2=1+1j
c3=c1+c2
print(c1,c2,c3)
s1="Python"
s2="Language"
s3=s1+s2
print(s1,s2,s3)
s4="10"
s5="20"
```

```
s6=s4+s5
print(s4,s5,s6)
```

Output

```
10 20 30
1.5 1.2 2.7
(1+2j) (1+1j) (2+3j)
Python Language PythonLanguage
10 20 1020
```

When arithmetic operation performed on two different data types, python always give result in broader type

1. Complex
2. Float
3. Int

Complex>float>int

Example:

```
# Write a program to add two integer numbers

a=int(input("Enter first integer value "))
b=int(input("Enter second integer value "))
c=a+b
print("sum of ",a,b,"is",c)
```

Output

```
Enter first integer value 10
Enter second integer value 20
sum of  10 20 is 30
```

eval()

eval stands for evaluate, this function evaluate string representation of expression and return value. This is a predefined function.

eval("10") → 10
eval("1.5") → 1.5
eval("1+2j") → 1+2j
eval("100+200") → 300

Example:

```
# Write a program to add two numbers

value1=eval(input("Enter Value1 "))
value2=eval(input("Enter Value2 "))
value3=value1+value2
print(value1,value2,value3,sep="\n")
```

Output

```
Enter Value1 1+2j
Enter Value2 5
(1+2j)
5
(6+2j)
```

Example:

```
>>> True+True
2
>>> True+False
1
>>> False+False
0
>>> 100+True
101
```

-operator (subtraction)

This operator is used to perform subtraction operation (OR) find the difference between two values. This operator is used with numeric type.

Example:

```
# Write a program to input sales of two years and find difference in sales

sales1=float(input("Enter First Year Sales "))
sales2=float(input("Enter Second Year Sales "))
diff_sales=sales1-sales2
print(sales1,sales2,diff_sales,sep="\n")
```

Output

```
Enter First Year Sales 40000
```

```
Enter Second Year Sales 60000
40000.0
60000.0
-20000.0
```

Example:

```
# Write a program to input two distances and calculate
difference

dist1=int(input("Enter Distance1 :"))
dist2=int(input("Enter Distance2 :"))
diff=dist1-dist2
print(dist1,dist2,diff,sep="\n")
```

Output

```
Enter Distance1 :300
Enter Distance2 :500
300
500
-200
```

***Operator**

This operator is used to perform two operations

1. Multiplying numbers
2. Repeating a sequence **n** times

If two operands are numeric type, it performs multiplication
If one operand is integer type and another is sequence type, it perform repeating

Example:

```
>>> a=10*5
>>> print(a)
50
>>> b=1.5*2
>>> print(b)
3.0
>>> c=1.5*1.5
>>> print(c)
2.25
```

```
>>> d=5*"NARESH"
>>> d
'NARESHNARESHNARESHNARESHNARESH'
>>> list1=[0]*100
>>> print(list1)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> sales_year=[0]*12
>>> print(sales_year)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> "ABC"*"XYZ"
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    "ABC"*"XYZ"
TypeError: can't multiply sequence by non-int of type 'str'
>>> "ABC"*1.5
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    "ABC"*1.5
TypeError: can't multiply sequence by non-int of type 'float'
```

Example:

```
# Write a program to input productname,qty,price and caluclate
total amount

#input
pname=input("Enter ProductName ")
qty=eval(input("Enter Qty "))
price=float(input("Enter Price "))

#process
total_amt=qty*price

#output
print("ProductName ",pname)
print("Qty ",qty)
print("Price ",price)
print("Total Amount ",total_amt)
```

Example

```
# Write a program to convert Dollar to Rs

d=int(input("Enter Amount in Dollar "))
rs=d*83
print(d,"$ is equal to Rs",rs)
```

Output

```
Enter Amount in Dollar 5
5 $ is equal to Rs 415
```

Example

```
# Write a program to find area of rectangle

l=float(input("Enter L value "))
b=float(input("Enter B value "))
a=l*b
print("Area of rectangle with ",l,b,"is",a)
```

Output

```
Enter L value 1.2
Enter B value 1.3
Area of rectangle with  1.2 1.3 is 1.56
```

Python support 3 division operators

1. / → float division
2. // → floor division
3. % → Modulo

Homework

Write a program to find area of circle
Write a program to input temp in C and convert into F
Write a program to input temp in F and Convert into C
Write a program input two numbers and find total and avg
Write a program input name,sub1,sub2 and calculate total,avg marks

Python support 3 division operators

1. / → float division
2. // → floor division
3. % → Modulo

/ → float division

This operator divides two numbers and return quotient.
This operator after dividing it returns result in float.
It is a binary operator and required two operands

```
>>> a=4/2
>>> print(a)
2.0
>>> print(type(a))
<class 'float'>
>>> b=5/2
>>> print(b,type(b))
2.5 <class 'float'>
>>> c=5/0
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    c=5/0
ZeroDivisionError: division by zero
>>> d=0/2
>>> print(d,type(d))
0.0 <class 'float'>
```

Example:

```
# Write a program to input two numbers and find total,avg

a=int(input("Enter first integer value :"))
b=int(input("Enter second integer value :"))
c=a+b
d=c/2
print(a,b,c,d,sep="\n")
```

Output

```
Enter first integer value :4
Enter second integer value :5
4
5
9
4.5
```

Example:

```
# Write a program to find simple intrest
# si=ptr/100

# input
p=float(input("Enter Amount "))
t=int(input("Enter Time "))
r=float(input("Enter Rate "))

# Process
si=p*t*r/100

# Output
print("Simple interest is ",si)
```

Output

```
Enter Amount 7000
Enter Time 16
Enter Rate 1.2
Simple interest is  1344.0
```

// → floor division

Return the floor of x, the largest integer less than or equal to x.
"x" is nothing but result

```
>>> x=5/2
>>> print(x,type(x))
2.5 <class 'float'>
>>> y=5//2
>>> print(y,type(y))
2 <class 'int'>
>>> z=-5//2
>>> print(z,type(z))
-3 <class 'int'>
```

Example:

```
# Write a program to remove last digit of input integer number
(+ve)

num=int(input("Enter any number "))
```

```
print("Before Deleting Last Digit ",num)
x=num//10
print("After Deleting Last Digit ",x)
```

Output

```
Enter any number 456789
Before Deleting Last Digit  456789
After Deleting Last Digit  45678
```

% Modular or Modulo Operator

This operator divides two numbers and returns remainder
It is binary operator and required two operands

Example:

```
>>> a=4%2
>>> print(a)
0
>>> b=5%3
>>> print(b)
2
>>> c=9%6
>>> print(c)
3
>>> d=9//3
>>> print(d)
3
>>> e=8//2
>>> print(e)
4
>>> f=8%2
>>> print(f)
0
```

Example:

```
# Write a program to print/read last digit of input number

num=int(input("Enter any integer number "))
last_digit=num%10
print(num,last_digit,sep="\n")
```

Output
Enter any integer number 146
146
6

**** power of operator or exponent operator**
This operator returns the power of input number.
It is a binary operator and required 2 operands.

Example:

```
>>> a=5**2
>>> print(a)
25
>>> b=1234**-1
>>> print(b)
0.0008103727714748784
>>> c=10**-1
>>> print(c)
0.1
>>> d=1000**-1
>>> print(d)
0.001
>>> d=1.5**2
>>> print(d)
2.25
```

Operator Precedence
Operator precedence define which order of execution of operators

The following table summarizes the operator precedence in Python, from highest precedence (most binding) to lowest precedence (least binding). Operators in the same box have the same precedence. Unless the syntax is explicitly given, operators are binary. Operators in the same box group left to right (except for exponentiation and conditional expressions, which group from right to left).

Note that comparisons, membership tests, and identity tests, all have the same precedence and have a left-to-right chaining feature as described in the Comparisons section.

Operator	Description
(expressions...), [expressions...], {key: value...}, {expressions...}	Binding or parenthesized expression, list display, dictionary display, set display
x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
await x	Await expression
**	Exponentiation
+x, -x, ~x	Positive, negative, bitwise NOT
*, @, /, //, %	Multiplication, matrix multiplication, division, floor division, remainder
+, -	Addition and subtraction
<<, >>	Shifts
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
not x	Boolean NOT
and	Boolean AND
or	Boolean OR
if – else	Conditional expression
lambda	Lambda expression
:=	Assignment expression

Relational Operators

These operators are used for comparing values.

Relational operators are binary operators, these operators required 2 operands

Operator	Description
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
!=	Not Equal
==	Equal

These operators can be applied on any type of data.
These operators after comparing, it returns Boolean value (True/False)
The expression which consists of relational operators is called Boolean expression.

Example:

```
10>5
True
10>20
False
>>> 10>=10
True
>>> 10>=5
True
>>> 10>=20
False
>>> 10>10
False
>>> 10>5>2
True
>>> 10>5>20
False
>>> 10>20>10
False
```

Example:

```
10<20
True
```

```
>>> 10<5
False
>>> 10<=10
True
>>> 10<=20
True
>>> 10<=5
False
>>> 10<20<30
True
>>> 10>5<2
False
>>> 10>5<6
True
```

Example:

```
>>> 10==10
True
>>> 10==20
False
>>> 10!=20
True
>>> 10!=10
False
```

Example:

```
>>> "A"=="A"
True
>>> "a">"A"
True
>>> "A"<"a"
True
```

How to find ASCII value of input character?

ord() function is used to find ascii value of input character

Example

```
ord('A')
65
ord('B')
66
```

```
ord('C')
67
ord('Z')
90
>>> ord('0')
48
>>> ord('9')
57
>>> ord('a')
97
>>> ord('z')
122
>>> '0'>'1'
False
>>> 'A'<'1'
False
>>> 10>=3
True
```

How to find character value of input ascii value?
chr() function returns character value of input ascii value

Example:

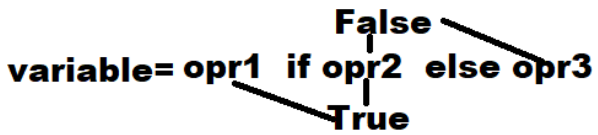
```
>>> chr(65)
'A'
>>> chr(90)
'Z'
>>> chr(48)
'0'
>>> chr(49)
'1'
>>> chr(97)
'a'
>>> chr(122)
'z'
```

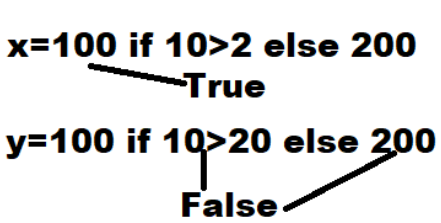
Conditional Operators

Conditional operator is ternary operator. This operator required 3 operands for performing operation.

The expression which uses conditional operator is called conditional expression.
Conditional expression evaluates expressions or statement based on condition.

Syntax:
variable-name=expression-1 if condition else expression2 (OR)
variable-name=opr1 if opr2 else opr3 (OR)
variable-name=true-oprr if Boolean-expr else false-opr





Example:
Write a program to find max of two integer numbers

```
# Input
a=int(input("Enter First Integer Value "))
b=int(input("Enter Second Integer Value "))

# Process
c=a if a>b else b

# Output
print(f'Maximum of {a} and {b} is {c}')
```

Output
Enter First Integer Value 20
Enter Second Integer Value 10
Maximum of 20 and 10 is 20

Example:

```
# Write a program to find input number is even or odd

# input
num=int(input("Enter integer number "))

# Process and Output
print(f"{num} is Even") if num%2==0 else print(f"{num} is Odd")
```

Output

```
Enter integer number 9
9 is Odd

Enter integer number 8
8 is Even
```

A conditional expression consists of one or more than one conditional operator.

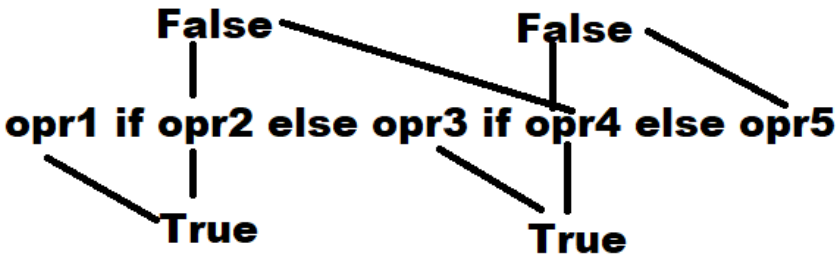
Example:

```
>>> 10 if True else 20
10
>>> 100 if False else 200
200
>>> 100 if 0 else 200
200
>>> 100 if 200 else 300
100
>>> "PYTHON" if 1 else "JAVA"
'PYTHON'
>>> 'PYTHON' if "A" else "JAVA"
'PYTHON'
```

Syntax:

Opr1 if opr2 else opr3 if opr4 else opr5

Multiple conditional operators are used to check multiple conditions or evaluating multiple Boolean expressions.



Example:

```
>>> 10 if True else 20 if True else 30
10
>>> 10 if False else 20 if True else 30
20
>>> 10 if False else 30 if False else 40
40
```

Logical Operators

Logical operators are used to combine two or more Boolean expressions or conditions.

Logical operators are represented using 3 keywords

1. and
2. or
3. not

and operator

Truth table "and" operator

Opr1	**Opr2**	**Opr1 and Opr2**
True	True	True
True	False	False
False	True	False
False	False	False

Example

```
# Login Application/Program

#input
uname=input("UserName :")
pwd=input("Password :")

# Process
```

```
print("Welcome") if uname=="nit" and pwd=="n123" else
print("Invalid username or password")
```

Output

UserName :nit
Password :abc
Invalid username or password

Example:

```
# Write a program to find max of three numbers

# input

a=int(input("Enter First Integer Value "))
b=int(input("Enter Second Integer Value "))
c=int(input("Enter Thrid Integer Value "))

# Process
res="equal" if a==b and a==c else a if a>=b and a>=c else b if
b>=a and b>=c else c

# Output
print(f"Max of {a},{b},{c} is {res}")
```

Example:

```
# write a program to input name and 2 subject marks and find
result(pass/fail)

name=input("Enter Name ")
sub1=int(input("Enter Subject1 Marks "))
sub2=int(input("Enter Subject2 Marks "))


print(f'Student Name {name}')
print(f'Student Subject1 Marks {sub1}')
print(f'Student Subject2 Marks {sub2}')

result="PASS" if sub1>=50 and sub2>=50 else "FAIL"

print(f'Student Result {result}')
```

Output
Enter Name suresh
Enter Subject1 Marks 40
Enter Subject2 Marks 99
Student Name suresh
Student Subject1 Marks 40
Student Subject2 Marks 99
Student Result FAIL

HW
Write a program to input name,sub1, sub2 and find grade based avg marks

>90 -→ A
>80<=90 → B+
>70<=80 → B
>60<=70 → C
<=60 → D

Write a program to calculate commission based sales
Sales>=60000 → 10%
Sales>=4000<60000 → 5%
Sales<40000 → 0

```
>>> True and True
True
>>> True and False
False
>>> 100 and 200
200
>>> 0 and 200
0
>>> 100 and 200 and 300
300
>>>100 and 0 and 300
0
>>> "Python" and "Java"
'Java'
```

```
>>> 1.5 and 2.5
2.5
>>> 1+2j and 1+3j
(1+3j)
```

Note: in and operator, PVM evaluates second operand if first operand is True. If first operand is False, PVM does not evaluate second operand, it return result of first operand.

```
name=input("Enter Name ")
sub1=int(input("Enter subject1 marks "))
sub2=int(input("Enter subject2 marks "))
avg=(sub1+sub2)/2

grade="A" if avg>90 else "B+" if avg>80 and avg<=90 else "B" if
avg>70 and avg<=80 else "C" if avg>60 and avg<=50 else "D"

print(f'Name {name}')
print(f'Subject1 Marks {sub1}')
print(f'Subject2 Marks {sub2}')
print(f'Grade {grade}')
```

Output

```
Enter Name kishore
Enter subject1 marks 45
Enter subject2 marks 47
Name kishore
Subject1 Marks 45
Subject2 Marks 47
Grade D
```

Example:

```
sales=int(input("Enter Sales :"))
comm=(sales*10/100) if (sales>=60000) else (sales*5/100) if
(sales>=40000 and sales<60000) else 0

print(f'Sales {sales}')
print(f'Comm {comm}')
```

Output

```
Enter Sales :75000
Sales 75000
Comm 7500.0


Enter Sales :45000
Sales 45000
Comm 2250.0

 Enter Sales :20000
Sales 20000
Comm 0
```

or operator

"or" keyword represents a logical "or" operator.
Truth table of or operator

Opr1	**Opr2**	**Opr1 or Opr2**
True	True	True
True	False	True
False	True	True
False	False	False

In or operation, if opr1 is True, PVM does not evaluate opr2 and returns the result of Opr1. If opr1 is False, PVM evaluate opr2 and return result of opr2

```
>>> 100 or 200
100
>>> 0 or 200
200
>>> 0 or 0
0
```

Example:

```
# Write a program to find input character is vowel or not

ch=input("Enter single character ")
```

```
print("vowel") if ch=='a' or ch=='e' or ch=='o' or ch=='i' or ch=='u' or
ch=='A' or ch=='E' or ch=='O' or ch=='U'  or ch=='I' else print("not
vowel")
```

Output

```
Enter single character X
not vowel

Enter single character a
Vowel
```

Example:

```
>>> 100 and 200 or 300
200
>>> 100 or 200 and 300
100
>>> 100 and 0 and 300 or 400
400
>>> 100 or 200 and 300 and 400 and 500
100
```

Example:

```
# Write a program  to input name,sub1,sub2 and find
result(pass/fail)

name=input("Enter Student Name ")
sub1=int(input("Enter Subject1 Marks "))
sub2=int(input("Enter Subject2 Marks "))

result="Fail" if sub1<40 or sub2<40 else "Pass"

print(f'Name {name}')
print(f'Subject1 Marks {sub1}')
print(f'Subject2 Marks {sub2}')
print(f'Result {result}')
```

Output

```
Enter Student Name suresh
Enter Subject1 Marks 30
Enter Subject2 Marks 99
```

Name suresh
Subject1 Marks 30
Subject2 Marks 99
Result Fail

Enter Student Name kishore
Enter Subject1 Marks 69
Enter Subject2 Marks 87
Name kishore
Subject1 Marks 69
Subject2 Marks 87
Result Pass

“not” operator

“not” is a keyword, which represents logical operator.
This operator is used with other operators to perform specific operations.

Truth table of “not” operator

Opr1	**not opr1**
True	False
False	True

Example

```
>>> not True
False
>>> not False
True
>>> not 0
True
>>> not 100
False
>>> not 100 and 200
False
>>> 100 and not 200
False
```

Membership Operator

This operator is used for searching given value inside collection of values. if value exists inside collection of values, this operator returns True else False

1. in
2. not in

It is binary operator and required 2 operands

Syntax: opr1 in **opr2**

Note: opr2 must be **collection type**

```
>>>10 in 10
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    10 in 10
TypeError: argument of type 'int' is not iterable
>>>10 in [10,20,30,40,50]
True
>>>100 in [10,20,30,40,50]
False
>>> "a" in "java"
True
>>> "a" in "python"
False
>>> "python" in "java oracle python"
True
>>> 10 not in [10,20,30,40,50]
False
>>> 10 not in [1,2,3,4,5]
True
```

Example:

```
# Write a program to find input character is vowel or not
ch=input("Enter any character ")
print("Vowel") if ch in "aeiouAEIOU"  else print("Not Vowel")
```

Output

```
Enter any character a
```

Vowel

Enter any character U
Vowel

Enter any character T
Not Vowel

Compound Assignment Operators OR Assignment Operators
Compound assignment operator performs two operations

1. binary operations
2. assignment

+=	
-=	
*=	
/=	
//=	
%=	
**=	
>>=	
<<=	
&=	
|=	
^=	

Example:

```
>>> a=10
>>> a++
SyntaxError: invalid syntax
>>> a--
SyntaxError: invalid syntax
>>> --a
10
>>> ++a
10
>>> ++++++a
10
>>> -+a
-10
```

Compound Assignment Operators OR Assignment Operators

Compound assignment operator performs two operations

3. binary operations
4. assignment

Compound assignment operators also called update operators.

+=	a=10
	print(a)
	10
	>>> a=a+5
	>>> print(a)
	15
	>>> a+=2
	>>> print(a)
	17
	>>> a+=1
	>>> print(a)
	18
	>>> a+=2+1
	>>> print(a)
	21

-=	>>> b=15
	>>> print(b)
	15
	>>> b-=1
	>>> print(b)
	14
	>>> b-=2
	>>> print(b)
	12

*=	>>> c=5
	>>> print(c)
	5
	>>> c*=2
	>>> print(c)
	10

/=	d=10 print(d) 10 d/=5

	print(d) 2.0

//=	>>> e=10
	>>> print(e)
	10
	>>> e//=3
	>>> print(e)
	3

%=	f=5 print(f) 5 f%=2 print(f) 1

**=	>>> a=5
	>>> print(a)
	5
	>>> a**=2
	>>> print(a)
	25

>>=	Right shift update A=0b1010 A=A>>2 (OR) A>>=2
<<=	Left shift update A=0b1010 A<<=2 → A=A<<2
&=	Bitwise and update A=0b1 B=0b0 A=A&B → A&=B
|=	Bitwise or update A=0b1 B=0b0 A=A|B → A|=B
^=	Bitwise xor update

	A=0b1 B=0b0 A=A^B → A^=B

Bitwise Operators
Bitwise operators are applied only on integer data type.

Bitwise Shift operators
These operators are used to shift given number of bits towards left or right side.

1. >> (Right shift)
2. << (left shift)

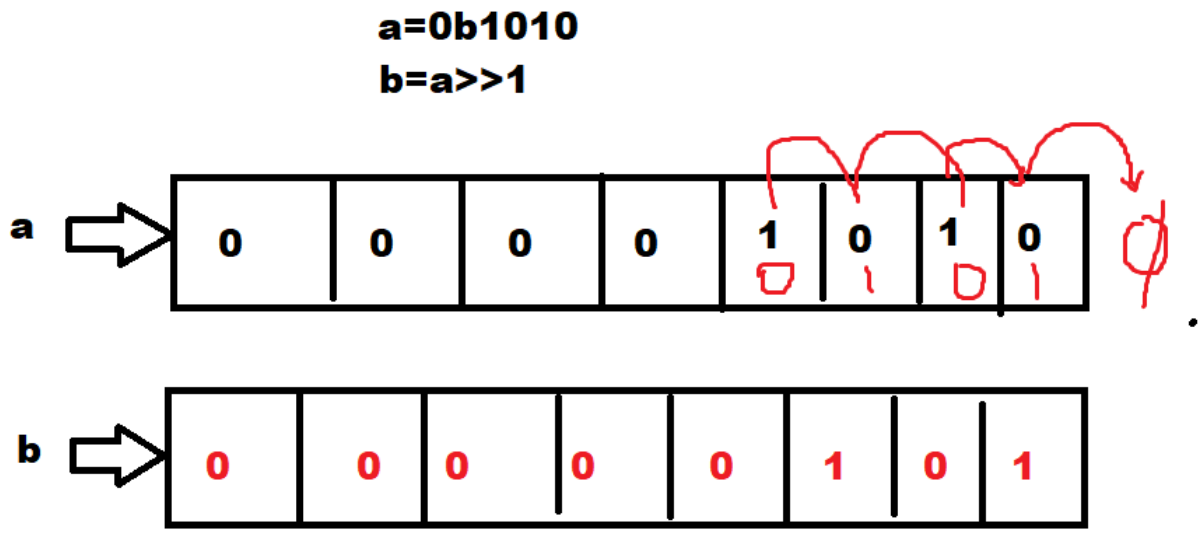
Applications of shift operators

1. Incrementing or decrementing value by adding or removing bits.
2. Memory Management
3. Encoding and decoding

>> right shift operator
This operator is used to shift given n bits toward right side

Syntax: opr>>n

It is a binary operator and required 2 operands.



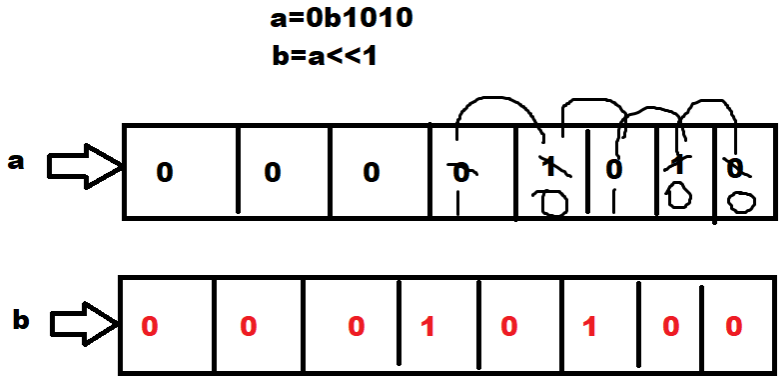
By shifting n bits towards right side the value get decremented.

```
>>> a=0b1010
>>> b=a>>1
>>> print(bin(a),bin(b))
0b1010 0b101
>>> print(a,b)
10 5
>>> x=0b101101
>>> y=x>>3
>>> print(bin(x),bin(y))
0b101101 0b101
>>> print(x,y)
45 5
>>> a=45
>>> b=a>>3
>>> print(a,b)
45 5
>>> print(bin(a),bin(b))
0b101101 0b101
```

Formula : opr//2 pow n → 45//2 pow 3 → 45//8 → 5

Left shift operator <<

This operator is used to shift n bits towards left side.
By shifting n bits towards left side the value get incremented
It is incremented by adding n bits at right side. (0's)




```
>>> a=10
>>> b=a<<2
>>> print(a,b)
10 40
```

```
>>> print(bin(a),bin(b))
0b1010 0b101000
>>> x=0b1010
>>> y=x<<2
>>> print(bin(x),bin(y))
0b1010 0b101000
>>> print(x,y)
10 40
```

Formula : opr* 2 pow n → 10*2pow2 → 10*4=40

Example:

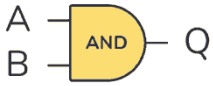
```
>>> x=0b1010
>>> y=x>>1<<1
>>> print(x,y)
10 10
>>> a=0b1011
>>> b=a>>1<<1
>>> print(a,b)
11 10
>>> print(bin(a),bin(b))
0b1011 0b1010
```

Other bitwise operators

Bitwise & (and) operator
Bitwise | (or) operator
Bitwise ^ (XOR) operator
Bitwise ~ not operator

These operators are used to apply logic gates while working embedded applications.

& operator
This operator is used to apply and gate

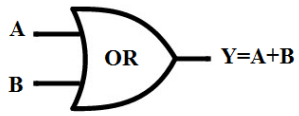


A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

```
>>> a=0b101
>>> b=0b110
>>> c=a&b
>>> print(bin(a),bin(b),bin(c))
0b101 0b110 0b100
>>> a=5
>>> b=6
>>> c=a&b
>>> print(bin(a),bin(b),bin(c))
0b101 0b110 0b100
>>> print(a,b,c)
5 6 4
>>> c=a and b
>>> print(a,b,c)
5 6 6
```

| bitwise or operator

This operator is used to apply or gate.



Inputs		Output
A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

```
>>> a=0b101
>>> b=0b110
>>> c=a|b
>>> print(bin(a),bin(b),bin(c))
0b101 0b110 0b111
>>> print(a,b,c)
5 6 7
```

```
>>> a=5
>>> b=6
>>> c=a&b
>>> print(bin(a),bin(b),bin(c))
0b101 0b110 0b100
>>> print(a,b,c)
5 6 4
>>> c=a|b
>>> print(a,b,c)
5 6 7
```

^ (XOR) Operator

This operator is used to apply XOR gate

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Example:

```
a=0b101
>>> b=0b110
>>> c=a^b
>>> print(bin(a),bin(b),bin(c))
0b101 0b110 0b11
>>> print(a,b,c)
5 6 3
```

Bitwise ~ not operator

It is unary operator and used one operand

Syntax: ~opr

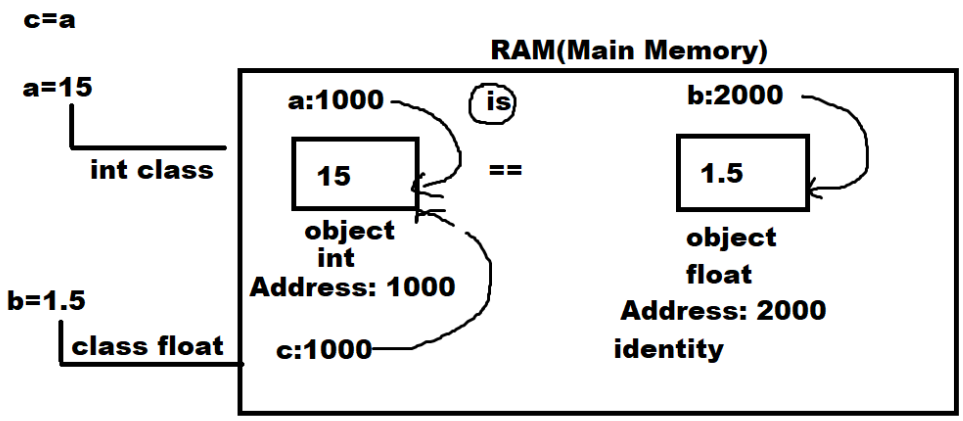
Formula : -(opr+1)

```
x=-6
y=~x
```

```
print(x,y)
-6 5
>>> a=0b101
>>> b=~a
>>> print(a,b)
5 -6
>>> print(bin(a),bin(b))
0b101 -0b110
```

Identity Operator

Python is an object oriented programming language.
Every data type in python is class and data is represented as objects
If python variables do not have value, it is having address of object created inside main memory.
Variables in python are called reference variables.



Identity operator is used to find two variables pointing to same object inside memory (OR) is used for comparing addresses.

id() : it is a predefined function which returns address/id of object.

```
>>> a=10
>>> id(a)
140718278048472
>>> b=1.5
>>> id(b)
2708510037616
>>> c=a
```

```
>>> id(c)
140718278048472
```

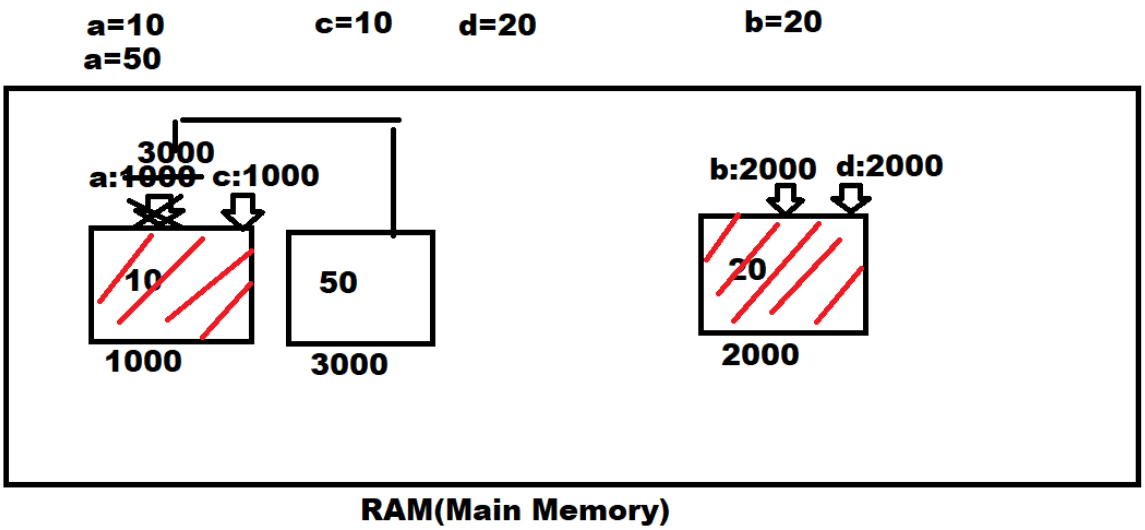
Python data types are classified into two categories

1. Mutable types
2. Immutable types

Mutable Data types:
These data types are used to create mutable objects, after creating object changes can be done.
Example: list,bytearray,set,dict

Immutable Data types:
These data types are used to create immutable objects, after creating these objects changes cannot do.
Example:
int,float,complex,bool,NoneType,tuple,string,range,bytes,frozenset

Immutable are sharable (one object is shared by number of variables)



Example
a=10

```
b=10
c=10
print(id(a),id(b),id(c))
x=1.5
y=1.5
print(id(x),id(y))

s1="nit"
s2="nit"
print(id(s1),id(s2))

list1=[10,20,30]
list2=[10,20,30]
print(id(list1),id(list2))
```

Output

```
140718278048472 140718278048472 140718278048472
2027071953072 2027071953072
2027077675664 2027077675664
2027077715264 2027077615360
```

Python support two identity operators

1. is
2. is not

is operator returns Boolean value, after comparing identity of two objects.

Example:

```
list1=[10,20]
list2=[10,20]
print(id(list1),id(list2))
print(list1==list2)
print(list1 is list2)
list3=list1
print(list1 is list3)
```

Output

```
2671875702208 2671832006272
True
False
```

True

What is difference between == and is operator in python?

The == operator compares the value or equality of two objects, whereas the Python is operator checks whether two variables point to the same object in memory.

Walrus operator

This operator is introduced in python 3.8 version.
Walrus operator is also called assignment expression operator.
:= walrus operator
This operator is used as part expression.

```
>>> a=10
>>> b=20
>>> c=(d=a+b)*(e=a-b)
SyntaxError: invalid syntax. Maybe you meant '==' or ':=' instead of '='?
>>> c=(d:=a+b)*(e:=a-b)
>>> print(c,d,e)
-300 30 -10
>>> a:=10
SyntaxError: invalid syntax
>>> c=(d:=5+2)-(e:=5-3)
>>> print(c,d,e)
5 7 2
```

Formatting output/formatting string

Format string is used for formatting output.
This formatting is done in different ways

1. Old style string formatting
2. New style string formatting
3. F-string (Python 3.8 version)

What is format string?
A string consists of replacement fields or formatting specifiers is called format string.

Old Style string formatting

Old style string formatting is nothing but, C-Style string formatting. In this formatting is done, the way it is done in C language.

Syntax
" formatting characters/replacement fields
"%(value,value,value,..)

%d → decimal integer
%o → Octal integer
%x → Hexadecimal integer
%f → float in fixed notation
%e → float in exponent notation
%s → String
%c → Character

Example:

```
rollno=125
name="naresh"
course="python"
fee=5000.0

print('''Rollno:%d
Name:%s
Course:%s
Fee:%.2f'''%(rollno,name,course,fee))
```

Output

```
Rollno:125
Name:naresh
Course:python
Fee:5000.00
```

Example:

```
a=65
print("%c"%(a))
print("%d"%(a))
print("%o"%(a))
print("%x"%(a))
b="naresh"
```

```
print("%s"%(b))
```

Output

```
A
65
101
41
Naresh
```

Example:

```
# Write a program to find area of circle
# area=pi*r*r
# Output: Area of circle with radius ---- is ----

r=float(input("Enter Radius of Circle "))
area=3.147*r*r

print("Area of circle with radius %.2f is %.2f"%(r,area))
```

Output

```
Enter Radius of Circle 1.2
Area of circle with radius 1.20 is 4.53
```

New style string formatting using format function

String data type provides a predefined function called format. This function is used for formatting string.

Syntax:

```
“replacement fields/place holders
“.format(variable,variable,variable)
```

These replacement fields are represented using curly braces {}
These replacement fields are replaced with values.

Example:

```
rollno=12
name="naresh"
course="python"
fee=5000.0
```

```
print('''Rollno : {}
Name : {}
Course : {}
Fee : {}'''.format(rollno,name,course,fee))
```

Output

```
Rollno : 12
Name : naresh
Course : python
Fee : 5000.0
```

Example:

```
a,b,c,d,e=10,20,30,40,50

print("{},{},{},{},{}".format(a,b,c,d,e))
# Fields identified with name
print("{p:},{q:},{r:},{s:},{t:}".format(p=a,q=b,r=c,s=d,t=e))
print("{p:},{q:},{r:},{s:},{t:}".format(t=e,p=c,q=b,r=d,s=a))
# Fields identified with position
print("{0:},{1:},{2:},{3:},{4:}".format(a,b,c,d,e))
print("{3:},{1:},{2:},{4:},{0:}".format(a,b,c,d,e))
```

Output

```
10,20,30,40,50
10,20,30,40,50
30,20,40,10,50
10,20,30,40,50
40,20,30,50,10
```

Formatting characters

d → decimal integer
o → octal integer
x → hexadecimal integer
b → binary integer
c → Character
s → string
f → float in fixed
e → float in expo

Example:

```
a=12

print("{:d},{:o},{:x},{:b}".format(a,a,a,a))
print("{p:d},{q:o},{r:x},{s:b}".format(p=a,q=a,r=a,s=a))

b=1.5
print("{:f},{:e}".format(b,b))
print("{:.2f},{:.2e}".format(b,b))
```

Output

```
12,14,c,1100
12,14,c,1100
1.500000,1.500000e+00
1.50,1.50e+00
```

f-string

f-string is introduced in python 3.8 version.
f-string is introduced in python 3.8 version.
A string prefix with f is called f-string.
f-string is having replacement fields, which are replaced with values.
Each replacement fields is represented using {} curly brace.

Syntax: f"{variable:type} {variable:type} {variable:type}"

d- decimal format
o → octal format
x → hexadecimal format
b → binary format
f → float fixed
e → float expo
s → string
c → character

Example:

```
a=10
b=20
c=30
```

```
print(f"The value of a={a},b={b},c={c}")
print(f"The value of a={a:d},b={b:d},c={c:d}")
print(f"The value of a={a:o},b={b:o},c={c:o}")
print(f"The value of a={a:x},b={b:x},c={c:x}")
print(f"The vlaue of a={a:b},b={b:b},c={c:b}")
print(f"Sum of {a},{b},{c} is {a+b+c}")

x=0b101
print(f'{x:b}')

f1=1.45
f2=1.45e-1
print(f"f1={f1:.2f},f2={f2:.2e}")

name="NARESH"
print(f"Name is {name}")

a=65
print(f"the value of a is {a:f}")
```

Output

```
The value of a=10,b=20,c=30
The value of a=10,b=20,c=30
The value of a=12,b=24,c=36
The value of a=a,b=14,c=1e
The vlaue of a=1010,b=10100,c=11110
Sum of 10,20,30 is 60
101
f1=1.45,f2=1.45e-01
Name is NARESH
the value of a is 65.000000
```

Control Statements

Conditional Control Statements

1. If
2. match

Looping Control Statements

1. while
2. for

Branching statements

1. break
2. continue
3. pass
4. return (functions)

Control statements are used to control the flow of execution of program.

Seq Alg	Selection Alg	Iterational Alg
1. a=int(input()) 2. b=int(input()) 3. c=a+b 4. print(c) 5. 6.	1. a=int(input()) 2. b=int(input()) 3. if a>b: 4. print(a) 5.else: 6. print(b) 7.	1.while/for: 2. print("Hello") 3. 4. 5.

Conditional control statements

Conditional control statements are used to execute block of statements based on condition.

Python support two types of conditional statements

1. if statement
2. match statement (python 3.10 version)

if statement

if is a conditional control statement.

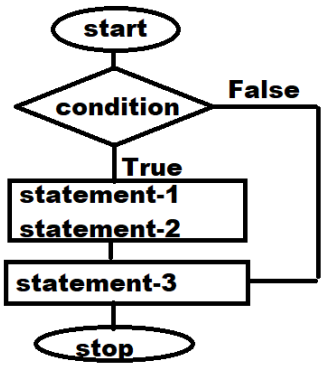
This statement is used to execute block of statements based on condition.

Types of if statements

1. simple if
2. if..else
3. if..elif..else (if..else ladder)
4. nested if

simple if

if without else is called simple if

Syntax:
If <condition>:
  Statement-1
  Statement-2
Statement-3
 	
If condition is True, PVM executes Statement-1,Statement-2 and statement-3
If condition is False, PVM executes Statement-3
	

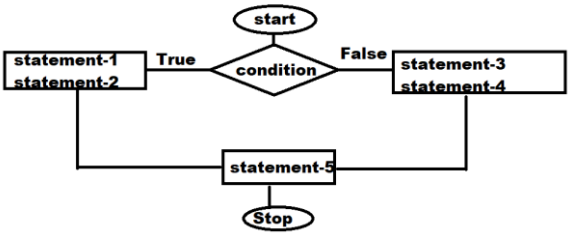
What is indentation in python?
The space given at beginning of the statement is called indentation.
Indentation is used for creating block.

Note: empty blocks are not allowed.

Example:
if 10>5:
  print("Hello")
 	
if 5>10:
  print("Bye")
 	
if True:
  print("Python")
 	
if False:
  print("Java")
 	
if 10>5:
  print("Oracle")
	Output
	Hello
	Python
	Oracle

If..else
This syntax is having two blocks.

1. if block
2. else block

Syntax:
if <condition>:
  statement-1
  statement-2
else:
  statement-3
  statement-4
 	
statement-5
 	
if condition is True, PVM executes statement-1,statement-2 and statement-5
if condition is False, PVM executes statement-3,statement-4 and statement-5
	

Example
if 10>5:
  print("Python")
else:
  print("Java")
 	
if 10>20:
  print("Python")
else:
  print("Java")
 	
if 10>5:
  print("Python")
else:
  print("Java")
 	
if 10>5:
  print("ABC")
	Output
	Python
	Java
	Python
	ABC
	PQR

else: print("XYZ") print("PQR")	

Example:

```
# Write a program to check whether a person is eligible to vote or not
# input age

name=input("Enter Name ")
age=int(input("Enter Age "))
if age>=18:
   print(f'{name} is eligible for vote')
else:
   print(f'{name} is not eligible for vote')

print("Thanks for using Voter Eligibility Application")
```

Output

```
Enter Name suresh
Enter Age 15
suresh is not eligible for vote
Thanks for using Voter Eligibility Application
```

 # Write a program to check whether a number entered by user is even or odd

```
num=int(input("Enter any number "))
if num%2==0:
   print(f'{num} is even')
else:
   print(f'{num} is odd')
```

Output

```
Enter any number 7
7 is odd

Enter any number 8
8 is even
```

Example:

```
# Write a program to check whether a number is divisible by 7 or
not

num=int(input("Enter any number "))
if num%7==0:
   print(f'{num} is divisible with 7')
else:
   print(f'{num} is not divisible with 7')
```

Output

```
Enter any number 21
21 is divisible with 7

Enter any number 27
27 is not divisible with 7
```

Example:

```
# Write a program to display "Hello" if a number entered by user is
a multiple of five
# otherwise "Bye"

num=int(input("Enter any number "))
if num%5==0:
   print("Hello")
else:
   print("Bye")
```

Output

```
Enter any number 15
Hello

Enter any number 12
Bye
```

Example:

```
# Write a program to check whether the last digit of number is
divisible by 3 or not
num=int(input("Enter any number "))
last_digit=num%10
if last_digit%3==0:
```

```
    print(f'{num} last digit is divisible with 3')
else:
    print(f'{num} last digit is not divisible with 3')
```

Output

```
Enter any number 126
126 last digit is divisible with 3
>>>
Enter any number 127
127 last digit is not divisible with 3
```

Example:

```
# Write a program to check a number entered is three digit
number or not

num=int(input("Enter any number "))
if  100<=num<=999:
    print("Three Digit Number ")
else:
    print("Not Three Digit Number ")
```

Output

```
Enter any number 99
Not Three Digit Number

Enter any number 125
Three Digit Number

Enter any number 1234
Not Three Digit Number
```

Example:

```
#Write a program to check whether a person is senior citizen or
not

age=int(input("Enter age of the person"))
if age>=60:
    print("Senior Citizen")
else:
```

```
print("Not Sernior Citizen")
```

Output

Enter age of the person100
Senior Citizen

Enter age of the person1000
Senior Citizen

Home Work

Q4. Write a program to find the lowest number out of two numbers excepted from user.

Q5. Write a program to find the largest number out of two numbers excepted from user.

Q8. Write a program to whether a number (accepted from user) is divisible by 2 and 3 both.

Q6. Write a program to check whether a number (accepted from user) is positive or negative.

Q1. Accept the temperature in degree Celsius of water and check whether it is boiling or not (boiling point of water in 100 °C.

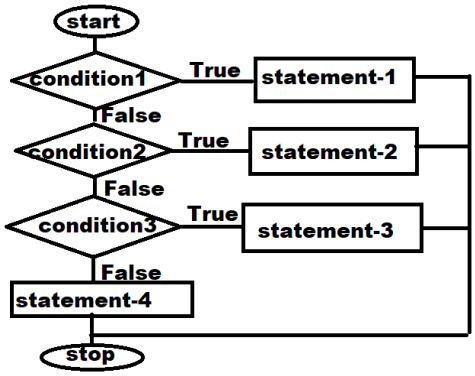
If..elif..else (if..else ladder)

This syntax is used for checking for multiple conditions.

Syntax:

```
If <codnition1>:
   Statement-1
elif <condition2>:
   Statement-2
elif <condition3>:
   Statement-3
else:
   Statement-4
```

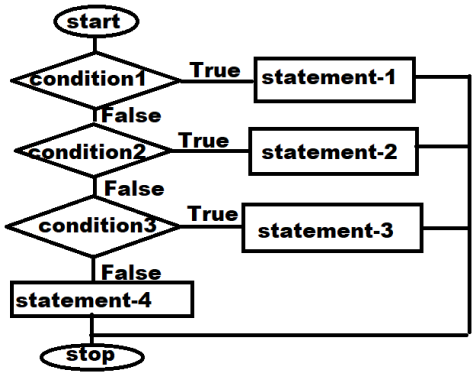
If condition1 is True, PVM executes statement-1
If condition1 is False and condition2 is True, PVM executes



statement-2 If condition1,condition2 are False and condition3 is True, PVM executes statement-3 If condition1,condition2,condition3 are False, PVM executes statement-4	

If..elif..else (if..else ladder)

This syntax is used for checking for multiple conditions.

Syntax:
If <codnition1>:
  Statement-1
elif <condition2>:
  Statement-2
elif <condition3>:
  Statement-3
else:
  Statement-4
 	
If condition1 is True, PVM executes statement-1
If condition1 is False and condition2 is True, PVM executes statement-2
If condition1,condition2 are False and condition3 is True, PVM executes statement-3
If condition1,condition2,condition3 are False, PVM executes statement-4
	

Example # Write a program to find input number is +ve,-ve or zero	Output Enter any number 5 +ve number

num=int(input("Enter any number "))
 if num>0:
 print("+ve number ")
 elif num<0:
 print("-ve number ")
 else:
 print("zero")
	Enter any number -7
	 -ve number
	
	 Enter any number 0
	 zero

# Write a program to find input character is alphabet,digit or special character 	
 	
 ch=input("Enter single charcter ")
 if (ch>='A' and ch<='Z') or (ch>='a' and ch<='z'):
 print(f'{ch} is alphabet')
 elif ch>'0' and ch<='9':
 print(f'{ch} is digit')
 else:
 print(f'{ch} special character')
	Output
	 Enter single charcter 6
	 6 is digit
	
	 Enter single charcter 8
	 8 is digit
	
	 Enter single charcter $
	 $ special character

https://www.hackerrank.com/challenges/py-if-else/problem?isFullScreen=true

```
n=int(input())
if n%2!=0:
    print("Weird")
elif n>=2 and n<=5:
    print("Not Weird")
elif n>=6 and n<=20:
    print("Weird")
elif n>20:
    print("Not Weird")
```

Q8. Write a program to calculate the electricity bill (accept number of unit from user) according to the following criteria :

Unit	Price
First 100 units	no charge
Next 100 units	Rs 5 per unit
After 200 units	Rs 10 per unit

(For example if input unit is 350 than total bill amount is Rs2000)

```
units=int(input("Enter Units :"))
```

```
if units<=100:
    amt=0
elif units>100 and units<=200:
    amt=(units-100)*5
elif units>200:
    amt=0+500+(units-200)*10

print(f'Total Amount {amt}')
```

Q1. Write a program to accept percentage from the user and display the grade according to the following criteria:

Marks	Grade
> 90	A
> 80 and <= 90	B
>= 60 and <= 80	C
below 60	D

```
p=float(input("Enter Percentage :"))
if p>90:
    print("A")
elif p>80 and p<=90:
    print("B")
elif p>=60 and p<=80:
    print("C")
elif p<60:
    print("D")
```

Q2. Write a program to accept the cost price of a bike and display the road tax to be paid according to the following criteria :

Cost price (in Rs)	Tax
> 100000	15 %
> 50000 and <= 100000	10%
<= 50000	5%

```
cost=float(input("Bike Cost :"))
if cost>100000:
    tax=cost*15/100
elif cost>50000 and cost<=100000:
    tax=cost*10/100
else:
    tax=cost*5/100

print(f'Bike Cost {cost}')
print(f'Road Tax {tax}')
```

Q1. Accept the following from the user and calculate the percentage of class attended:

a. Total number of working days

b. Total number of days for absent

After calculating percentage show that, If the percentage is less than 75, than student will not be able to sit in exam.

```
w_days=int(input("Total number of working days :"))
a_days=int(input("Total number of day  for absent :"))

p=(w_days-a_days)/w_days*100

if p<75:
    print("your not elg to sit in exam")
else:
    print("your elg to sit in exam")
```

Output

```
Total number of working days :30
Total number of day  for absent :5
your elg to sit in exam

Total number of working days :30
Total number of day  for absent :10
your not elg to sit in exam
```

Q3. A company decided to give bonus to employee according to following criteria:

Time period of Service	Bonus
More than 10 years	10%
>=6 and <=10	8%
Less than 6 years	5%

Ask user for their salary and years of service and print the net bonus amount.

```
salary=float(input("Enter Salary "))
service=int(input("Enter Service "))
if service>10:
```

```
    bonus=salary*10/100
elif service>=6 and serivce<=10:
    bonus=salary*8/100
else:
    bonus=salary*5/100

print(f'Salary {salary}')
print(f'Serivce {service}')
print(f'Bonus {bonus}')
print(f'Net Bonus {salary+bonus}')
```

Q4. Accept the marked price from the user and calculate the Net amount as(Marked Price – Discount) to pay according to following criteria:

Marked Price	Discount
>10000	20%
>7000 and <=10000	15%
<=7000	10%

```
price=float(input("Enter Price "))
if price>10000:
    dis=price*20/100
elif price>7000 and price<=10000:
    dis=price*15/100
else:
    dis=price*10/100

print(f'Net Amount is {price-dis}')
```

Nested if

If within if is called nested if

If followed by if is called nested if

Syntax:

```
if <condition1>:
   if <condition2>:
      statement-1
  else:
      statement-2
else:
   statement-3
```

if condition1,condition2 are True, PVM executes statement-1 if condition1 True but condition2 is False, PVM executes Statement-2 if condition1 is False, PVM executes statement-3

Example:

```
# Login application

user=input("UserName :")
if user=="nit":
    pwd=input("Password :")
    if pwd=="n123":
        print("welcome")
    else:
        print("invalid password")
else:
    print("invalid username")
```

Output

```
UserName :nit
Password :n112
invalid password

Password :n123
welcome
```

Example

```
# Write a program to find max of 3 numbers (without using and operator)

a=int(input("Enter value of a"))
b=int(input("Enter value of b"))
c=int(input("Enter value of c"))
if a>b:
    if a>c:
        print(f'{a} is max')
    else:
        print(f'{c} is max')
elif b>a:
    if b>c:
```

```
        print(f'{b} is max')
    else:
        print(f'{c} is max')
elif c>a:
    if c>b:
        print(f'{c} is max')
    else:
        print(f'{b} is max')
else:
    print("equal")
```

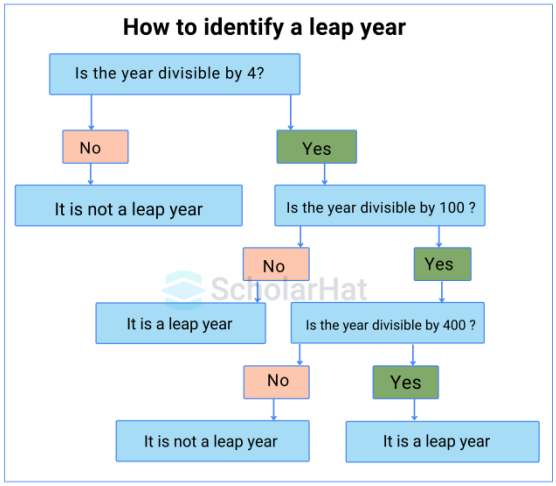
Output

```
Enter value of a10
Enter value of b20
Enter value of c10
20 is max

Enter value of a10
Enter value of b10
Enter value of c10
Equal
```

```
# Write a program to find input year is leap or not

year=int(input("Enter Year "))
if year%4==0:
    if year%100==0:
        if year%400==0:
            print("Leap Year")
        else:
            print("Not Leap Year")
    else:
        print("Leap year")
else:
    print("Not Leap Year")
```



Example:

```
# Write a program to input 2 subject marks and find result pass/fail

sub1=int(input("Subject1 Marks :"))
sub2=int(input("Subject2 Marks :"))

if sub1>=40:
   if sub2>=40:
      print("pass")
   else:
      print("sub2 Fail")
else:
   print("sub1 Fail")
```

Output

```
Subject1 Marks :90
Subject2 Marks :89
pass

Subject1 Marks :99
Subject2 Marks :30
sub2 Fail

Subject1 Marks :30
Subject2 Marks :99
sub1 Fail
```

Example:

```
# Write a program input an alphabet
# if input alphabet is in uppercase convert into lowercase
# if input alphabet is in lowercase convert into uppercase

ch=input("Enter Alphabet ")

if ch>='a' and ch<='z':
  print(f'Uppercase {chr(ord(ch)-32)}')
elif ch>='A' and ch<='Z':
   print(f'Lowercase {chr(ord(ch)+32)}')
else:
   print("input character is not alphabet")
```

Output
Enter Alphabet r
Uppercase R

Enter Alphabet R
Lowercase r

Enter Alphabet *
input character is not alphabet

pass

How to define empty blocks in python?
In python empty blocks are defined by including "pass" statement or keyword.

pass is a null operation — when it is executed, nothing happens. It is useful as a placeholder when a statement is required syntactically, but no code needs to be executed,

Example:
```
if 10>2:
   pass

print("hello")
print("bye")
```

Output
hello
bye

hello
bye

match

match

match statement is introduced in python 3.10 version.

Match statement is similar to switch..case in other languages (C,C++ and Java)
Match statement is called selection statement, which execute block of statements based condition.

Syntax:

```
match <parameter>:
   case pattern-1:
      code for pattern-1
  case pattern-2:
       code for pattern-2
  case pattern-3:
       code for pattern-3
  case _:
       default case
```

match followed by parameter to be matched with patterns.
If parameter value match with any pattern, execute code for pattern . if parameter values does not match with any pattern, it execute default case.

1. Simple match
2. Match with or operator
3. Match with if condition

Example:

```
a=3

match a:
   case 1:
      print("one")
   case 2:
      print("two")
   case 3:
      print("three")
   case _:
      print("default")
```

Output

Three

Example:

```
print("***MENU****")
print("1.Area of Circle ")
print("2.Area of Triangle")
print("3.Exit")
opt=int(input("Enter Your Option "))
match opt:
   case 1:
      r=float(input("Enter Radius of Circle "))
      a=3.147*r*r
      print(f'Area of circle is {a:.2f}')
   case 2:
      base=float(input("Enter Base of Triangle "))
      height=float(input("Enter Height of Triangle "))
      a=0.5*base*height
      print(f'Area of triangle is {a:.2f}')
   case 3:
      print("Thanks for using Menu...")
   case _:
      print("Please enter option from 1-3")
```

Output

```
***MENU****
1.Area of Circle
2.Area of Triangle
3.Exit
Enter Your Option 1
Enter Radius of Circle 1.5
Area of circle is 7.08

***MENU****
1.Area of Circle
2.Area of Triangle
3.Exit
Enter Your Option 2
Enter Base of Triangle 1.2
Enter Height of Triangle 1.3
Area of triangle is 0.78
```

Case with multiple patterns using or operator

Multiple patterns are separated with | (or) symbol

Example:

```
a=8
match a:
   case 1|3:
      print("1 or 3")
   case 5|7|12:
      print("5 or 7 or 12")
   case 4|8|16:
      print("4 or 8 or 16")
   case _:
      print("does not match any value/pattern")
```

Output

4 or 8 or 16

Example:

```
ch=input("Enter any character")
match ch:
   case 'a'|'e'|'i'|'o'|'u':
      print("Vowel")
   case _:
      print("not vowel")
```

Output

Enter any charactera
Vowel

Enter any charactero
Vowel

Enter any characterx
not vowel

match..case statement with if

Example:

```
num=int(input("Enter any number "))
match num:
   case num if num>0 :
```

```
        print("+ve num")
    case num if num<0:
        print("-ve number")
    case _:
        print("zero")
```

Output

Enter any number 7
+ve num

Enter any number 0
zero

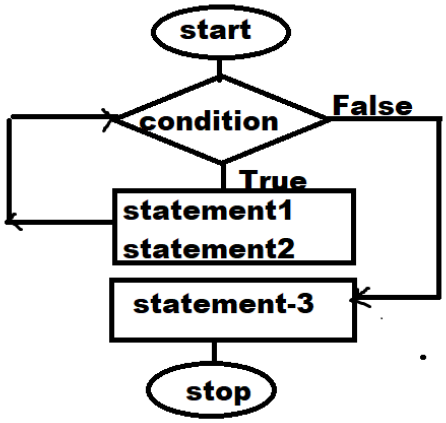
Looping Control statements

Looping control statements are used to repeat block of statements number of times or until given condition.
Python support 2 looping control statements

1. While loop
2. For loop

While loop

"while" keyword represents "while" loop
While loop repeat statements until given condition.

Syntax: while <condition>: statement-1 statement-2 statement-3 while statement repeat statement-1,statement-2 until given condition is True. If condition is False, stop repeating statement-1 and statement-2 and continue with statement-3	

While loop required 3 statements

1. Initialization statement
2. Condition
3. Update statement

Example:	**Output**

a=1
while a<=5:
  print("Hello")
  a=a+1
	Hello
	Hello
	Hello
	Hello
	Hello

# Write a program to print 1 2 3 4 5 using while loop 	
 	
num=1
while num<=5:
  print(num,end=' ')
  num=num+1
 	
print()
# Write a program to print 5 4 3 2 1 using while loop 	
num=5
while num>=1:
  print(num,end=' ')
  num-=1
 	
print()
# Write a program 1 4 9 16 25 using while loop 	
num=1
while num<=5:
  print(num2,end=' ')**
  num+=1
 	
print()
# Write a program to print 25 16 9 4 1 using while loop 	
num=5
while num>=1:
  print(num2,end=' ')**
	Outputs
	
	1 2 3 4 5
	5 4 3 2 1
	1 4 9 16 25
	25 16 9 4 1
	A B C D E

num-=1 **print()** **# Write a program print A B C D E using while loop** **n=65** **while n<=69:** **print(chr(n),end=' ')** **n=n+1**	

# Write a program to print sum of 10 numbers **# input 10 numbers from keyboard** a=1 s=0 while a<=10: num=int(input("Enter any number ")) s=s+num a=a+1 avg=s/10 print(f'Sum is {s}') print(f'Avg is {avg}')	**Output** Enter any number 1 Enter any number 2 Enter any number 3 Enter any number 4 Enter any number 5 Enter any number 6 Enter any number 7 Enter any number 8 Enter any number 9 Enter any number 10 Sum is 55 Avg is 5.5
# Write a program to print table for input number num=int(input("Enter any number ")) i=1 while i<=10: p=num*i print(f'{num}x{i}={p}') i=i+1	**Output** Enter any number 6 6x1=6 6x2=12 6x3=18 6x4=24 6x5=30 6x6=36 6x7=42 6x8=48 6x9=54

	6x10=60

# Write a program to find length of number 	
 	
num=int(input("Enter any number "))
c=0
while num>0:
  c=c+1
  num//=10
 	
print(f"Length of Count of Digits {c}")
	Output
	Enter any number 45
	Length of Count of Digits 2
	
	Enter any number 98654
	Length of Count of Digits 5

# Write a program to print sum of digits of input number 	
 	
num=int(input("Enter any number "))
 	
s=0
while num>0:
  r=num%10
  s=s+r
  num=num//10
 	
print(f'sum of digits is {s}')
	Enter any number 7592
	sum of digits is 23
	
	Enter any number 2345
	sum of digits is 14

# write a program to count even digits of input numbers 	
# 1346 --> 2 	
# 135 --> 0 	
 	
num=int(input("Enter any number "))
c=0
while num>0:
  r=num%10
  if r%2==0:
  c=c+1
  num=num//10
	Output
	Enter any number 139
	Count of even digits 0

print(f'Count of even digits {c}')	

# write a program to count even and odd digits of input numbers 	
 	
num=int(input("Enter any number "))
c1=0
c2=0
while num>0:
  r=num%10
  if r%2==0:
  c1=c1+1
  else:
  c2=c2+1
  num=num//10
 	
print(f'Count of even digits {c1}')
print(f'Count of odd digits {c2}')
	Output
	Enter any number 1369
	Count of even digits 1
	Count of odd digits 3

# Write a program input decimal number and convert into binary string 	
 	
num=int(input("Enter any number "))
binary=''
 	
while num!=0 and num!=1:
  r=num%2
  binary=binary+str(r)
  num=num//2
 	
binary=binary+str(num)
 	
print(binary[::-1]) #Slicing Operation
	Output
	Enter any number 27
	11011
	
	Enter any number 8
	1000

Home Work

1. **Write a program to convert decimal to octal**
2. **Write a program to convert decimal to hexadecimal**

Example:

```
# Decimal to Hexadecimal

num=int(input("Enter any number "))
s1=''
while num>0:
    r=num%16
    if r==10:
        r='a'
    elif r==11:
        r='b'
    elif r==12:
        r='c'
    elif r==13:
        r='d'
    elif r==14:
        r='e'
    elif r==15:
        r='f'
    s1=s1+str(r)
    num=num//16

s1=s1[::-1]
s1="0x"+s1
print(s1)
```

Output

```
Enter any number 26
0x1a
```

Example:

```
# Write a program to reverse a number

num=int(input("Enter any number "))
rev=0
```

```
while num>0:
    r=num%10
    rev=(rev*10)+r
    num=num//10

print(rev)
```

Output
Enter any number 123
321

Write a program to find input number is palindrome or not

```
num=int(input("Enter any number "))
rev=0
num1=num

while num>0:
    r=num%10
    rev=(rev*10)+r
    num=num//10

if rev==num1:
    print("pal")
else:
    print("not pal")
```

Output
Enter any number 121
pal

Enter any number 123
not pal

Example:
Write a program to find input number is armstrong or not

In number theory, a narcissistic number in a given number
#base is a number that is the

#sum of its own digits each raised to the power of the number of digits

```
# example : 153
# length 3
# 1 pow 3 + 5 pow 3 + 3 pow 3 = 153
# 153 == 153

num=int(input("Enter any number between 100 to 999"))
num1=num
s=0
while num>0:
    d=num%10
    s=s+(d**3)
    num=num//10

if num1==s:
    print("armstrong number")
else:
    print("not armstrong number")
```

Output

```
Enter any number between 100 to 999   153
armstrong number

Enter any number between 100 to 999   370
armstrong number

Enter any number between 100 to 999  654
not armstrong number
```

 # Write a program to find input number is prime or not

```
num=int(input("Enter any number "))
c=0
i=1
while i<=num:
    r=num%i
    if r==0:
```

```
    c=c+1
  i=i+1

if c==2:
  print(f'{num} is prime')
else:
  print(f'{num} is not prime')
```

for loop

"for" is a keyword which represents for loop in python.
The for statement is used to iterate over the elements of a sequence (such as a string, tuple or list) or other iterable object:

Syntax:

```
for variable in collection/iterable:
    statement-1
    statement-2
```

for loop each time read value generated by iterable
after reading value, it execute block of statements

Example:	**Output**
	Hello
for x in "123456":	**Hello**
print("Hello")	**Hello**
	Hello
for y in "python":	**Hello**
print("naresh")	**Hello**
	naresh
for z in "nit":	**naresh**
print("PYTHON")	**naresh**
	naresh
for p in "":	**naresh**
print("NIT")	**naresh**
	PYTHON
for a in [10,20,30,40,50]:	**PYTHON**
print("PY")	**PYTHON**
	PY

for b in "PYTHON":
  print(b)
 	
for x in [10,20,30,40,50]:
  print(x)
	PY
	PY
	PY
	PY
	P
	Y
	T
	H
	O
	N
	10
	20
	30
	40
	50

range data type

The range type represents an immutable sequence of numbers and is commonly used for looping a specific number of times in for loops.

Range data type is used to generate sequence of integers

Range data type required 3 inputs

1. Start : starting value of the range
2. Stop (not included) : end of value of range
3. Step : difference between values within range (increment/decrement value)

Syntax-1: range(stop)

Syntax-2: range(start,stop,[step])

Syntax-1: range(stop)

This syntax create range with default start and stop values

Example: range(10) → start=0,stop=10,step=+1

Default start is 0 and step is +1

This syntax generate only +ve range of integer values.

Example:

```
for x in range(5):
    print(x)

for y in range(10):
    print(y,end=' ')

for z in range(-5):
    print(z)

print()
for k in range(5):
    print("Hello")
```

Output

```
0
1
2
3
4
0 1 2 3 4 5 6 7 8 9
Hello
Hello
Hello
Hello
Hello
```

Syntax-2: range(start,stop,[step])

This syntax allows to generate +ve sequence of integers and –ve sequence integers in increment order and decrement order.
If step is not given, it default to +1

If step is +ve, start<stop
If step is –ve, start>stop

Example:

```
# Generating integers from 1 to 10
for x in range(1,11):
    print(x,end=' ')
```

```
print()
# Generating integers from 10 to 1
for x in range(10,0,-1):
    print(x,end=' ')

print()
# Generate integers from -1 to -10
for x in range(-1,-11,-1):
    print(x,end=' ')

print()
# Genrate integers from -10 to -1
for x in range(-10,0,1):
    print(x,end=' ')

print()
# generate integers from 5 to -5
for x in range(5,-6,-1):
    print(x,end=' ')

print()
# geneate integers from -5 to 5
for x in range(-5,6,1):
    print(x,end=' ')

print()
# generate even integers 2 4 6 8 10 12 14 16 18 20
for x in range(2,21,2):
    print(x,end=' ')

print()
# generate odd integers 1 3 5 7 9 11 13 15 17 19
for x in range(1,20,2):
    print(x,end=' ')

print()

# generate the following series 1 4 9 16 25 .. 100
```

```
for x in range(1,11):
    print(x**2,end=' ')
```

Output

```
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
-1 -2 -3 -4 -5 -6 -7 -8 -9 -10
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1
5 4 3 2 1 0 -1 -2 -3 -4 -5
-5 -4 -3 -2 -1 0 1 2 3 4 5
2 4 6 8 10 12 14 16 18 20
1 3 5 7 9 11 13 15 17 19
1 4 9 16 25 36 49 64 81 100
```

Example:

```
# Write a program to math table of input number

num=int(input("Enter any number ")) # 5

for i in range(1,11): # 1 2 3 4 5 6 7 8 9 10
    p=num*i
    print(f'{num}x{i}={p}')
```

Output

```
Enter any number 9
9x1=9
9x2=18
9x3=27
9x4=36
9x5=45
9x6=54
9x7=63
9x8=72
9x9=81
9x10=90
```

Example:

```
# Write a program to print the following charcter series
```

```
# A B C D E F G H .... Z

for var in range(65,91):
    print(chr(var),end=' ')

print()
# a b c d e f g  h  .... z
for var in range(97,123):
    print(chr(var),end=' ')
```

Output

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z

Note:

Range required integer inputs
Range step value should not be zero

Example:

```
for var in range(10,1):
    print(var)

for var in range(1,10,-1):
    print(var)

for var in range(-1,-10):
    print(var)

for var in range(-10,-1,-1):
    print(var)

for var in range(ord('A'),ord('Z')):
    print(chr(var),end=' ')

print()
```

Output

A B C D E F G H I J K L M N O P Q R S T U V W X Y

Example:	**Output**

# Write a program to input n numbers and print sum,avg 	
 	
n=int(input("Enter value of n"))
 	
s=0
for i in range(n): # start=0,stop=n,step=1
  num=int(input("Enter any number "))
  s=s+num
 	
print(f'Sum is {s}')
print(f'Avg is {s/n:.2f}')
	Enter value of n10
	Enter any number 1
	Enter any number 2
	Enter any number 3
	Enter any number 4
	Enter any number 5
	Enter any number 6
	Enter any number 7
	Enter any number 8
	Enter any number 9
	Enter any number 10
	Sum is 55
	Avg is 5.50

# Wrie a program to find the sum of the following series $1^2+2^2+3^2+4^2+5^2+\dots+n^2$ **Example** # Wrie a program to find the sum of the following series n=int(input("Enter value of n")) s=0 for i in range(1,n+1): s=s+(i**2) print(f'sum of series is {s}')	**Output** Enter value of n5 sum of series is 55
# Wrie a program to find the sum of the following series $1^1+2^2+3^3+4^4+5^5+\dots+n^n$ n=int(input("Enter value of n")) s=0 for i in range(1,n+1): s=s+(i**i) print(f'sum of series is {s}')	**Output** Enter value of n4 sum of series is 288
Fibonacci series **0 1 1 2 3 5 8 13 21**	**Output** Enter how many terms?8

n=int(input("Enter how many terms?"))
a=0
b=1
print(a,b,end=' ')
for i in range(n-2):
  c=a+b
  print(c,end=' ')
  a=b
  b=c
	0 1 1 2 3 5 8 13

# Write a program to find factorial of input number num=int(input("Enter any number ")) fact=1 for i in range(1,num+1): fact=fact*i print(f'Factorial of {num} is {fact}')	**Output** Enter any number 3 Factorial of 3 is 6 Enter any number 0 Factorial of 0 is 1

# Number of 1 Bits 	
# Write a program that takes the binary representation of a positive integer and returns the number of 	
# set bits it has (also known as the Hamming weight). 	
# Input: n = 11 	
# Output: 3 	
# Explanation: 	
 	
# The input binary string 1011 has a total of three set bits. 	
 	
num=int(input("Enter any number "))
c=0
while num>0:
  b=num%2
  if b==1:
  c=c+1
  num=num//2	

print(c)	

Nested Looping Statements
Defining looping statement within looping statement is called nested looping statements.

1. Nested while
2. Nested for

Nested for
For loop inside for loop is called nested for loop.

Syntax:
Nested Looping Statements
Defining looping statement within looping statement is called nested looping statements.

1. Nested while
2. Nested fro

Nested for
For loop inside for loop is called nested for loop.

Syntax:
for <variable-name> in <iterable>: → Outer for loop
for <variable-name> in <iterable>: → Inner for loop
statement-1
statement-2
statement-3

Example:	**Output**
# Write a program to print tables from 1 to 10 for num in range(1,11): # Outer Loop for i in range(1,11): # Inner Loop	**1x1=1** **1x2=2** **1x3=3** **1x4=4** **1x5=5** **1x6=6** **1x7=7** **1x8=8**

print(f'{num}x{i}={num*i}') input()	**1x9=9** **1x10=10** **.....**

Example
# Write a program to generate prime numbers from 2 to 30 	
 	
for num in range(2,31):
c=0
for i in range(1,num+1):
if num%i==0:
c+=1
if c==2:
print(num,end=' ')
	Output
	2 3 5 7 11 13 17 19 23 29

Example
# Write a program to generate factorial of all the numbers from 1 to 5 	
 	
for num in range(1,6):
fact=1
for i in range(1,num+1):
fact=fact*i
print(f'{num}--->{fact}')
	Output
	1--->1
	2--->2
	3--->6
	4--->24
	5--->120

```
1 1 1 1 1
```

```
for i in range(1,6):
  print("1",end=' ')
```

```
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
```

```
for i in range(1,6):    row
  for j in range(1,6):   col
    print("1",end=' ')   value
  print()
```

```
1
1
1
1
1
```

```
for i in range(1,6):
   print("1")
```

	1	2	3	4	5
1	1	1	1	1	1
2	2	2	2	2	2
3	3	3	3	3	3
4	4	4	4	4	4
5	5	5	5	5	5

```
for i in range(1,6):
  for j in range(1,6):
    print(i,end=' ')
  print()
```

	1	2	3	4	5
5	5	5	5	5	5
4	4	4	4	4	4
3	3	3	3	3	3
2	2	2	2	2	2
1	1	1	1	1	1

```
for i in range(5,0,-1):
  for j in range(1,6):
    print(i,end=' ')
  print()
```

	1	2	3	4	5
1	1	1	1	1	1
2	0	0	0	0	0
3	1	1	1	1	1
4	0	0	0	0	0
5	1	1	1	1	1

```
for i in range(1,6):
  for j in range(1,6):
    if i%2==0:
      print("0",end=' ')
    else:
      print("1",end=' ')
  print()
```

	1	2	3	4	5
1	1	2	3	4	5
2	1	2	3	4	5
3	1	2	3	4	5
4	1	2	3	4	5
5	1	2	3	4	5

```
for i in range(1,6):
  for j in range(1,6):
    print(j,end=' ')
  print()
```

	5	4	3	2	1
1	5	4	3	2	1
2	5	4	3	2	1
3	5	4	3	2	1
4	5	4	3	2	1
5	5	4	3	2	1

```
for i in range(1,6):
  for j in range(5,0,-1):
    print(j,end=' ')
  print()
```

```
5 5 5 5 5
4 4 4 4
3 3 3
2 2
1
```

```
for i in range(5,0,-1):
  for j in range(1,i+1):
    print(i,end=' ')
  print()
```

	1	2	3	4	5
1	1				
2	2	2			
3	3	3	3		
4	4	4	4	4	
5	5	5	5	5	5

```
for i in range(1,6):
  for j in range(1,i+1):
    print(i,end=' ')
  print()
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
for i in range(1,6):
  for j in range(1,i+1):
    print(j,end=' ')
  print()
```

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
for i in range(5,0,-1):
  for j in range(1,i+1):
     print(j,end=' ')
  print()
```

	5	4	3	2	1
1					1
2				2	2
3			3	3	3
4		4	4	4	4
5	5	5	5	5	5

```
for i in range(1,6):
   for j in range(5,0,-1):
      if i>=j:
         print(i,end=' ')
      else:
         print("  ",end=' ')
   print()
```

	5	4	3	2	1
1					1
2				2	1
3			3	2	1
4		4	3	2	1
5	5	4	3	2	1

```
for i in range(1,6):
   for j in range(5,0,-1):
      if i>=j:
         print(j,end=' ')
      else:
         print('  ',end=' ')
   print()
```

	1	2	3	4	5
1	1	2	3	4	5
2		2	3	4	5
3			3	4	5
4				4	5
5					5

```
for i in range(1,6):
   for j in range(1,6):
      if j>=i:
         print(j,end=' ')
      else:
         print('  ',end=' ')
   print()
```

	5	4	3	2	1
5	5	4	3	2	1
4		4	3	2	1
3			3	2	1
2				2	1
1					1

```
for i in range(5,0,-1):
   for j in range(5,0,-1):
      if j<=i:
         print(j,end=' ')
      else:
         print(' ',end= ' ')
   print()
```

```
   65 66 67 68 69
1  A B C D E
2  A B C D
3  A B C
4  A B
5  A
```

```
for i in range(1,6):
  for j in range(65,71-i):
      print(chr(j),end=' ')
  print()
```

```
   65 66 67 68 69
69
68
67
66
65
```

```
for i in range(69,64,-1):
  for j in range(65,i+1):
    print(chr(j),end=' ')
  print()
```

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

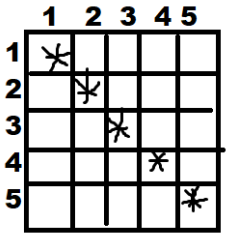
```
num=1
for i in range(1,6):
  for j in range(1,i+1):
      print(num,end= ' ')
      num=num+1
  print()
```

```
A
B C
D E F
G H I J
K L M N O
```

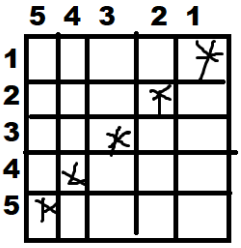
```
num=65
for i in range(1,6):
  for j in range(1,i+1):
    print(chr(num),end=' ')
    num=num+1
  print()
```

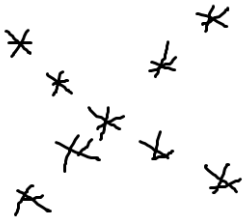
```
        A
      B C
    D E F
  G H I J
K L M N O
```

```
num=65
for i in range(1,6):
  for j in range(5,0,-1):
    if i>=j:
      print(chr(num),end=' ')
    else:
      print('  ',end=' ')
  print()
```



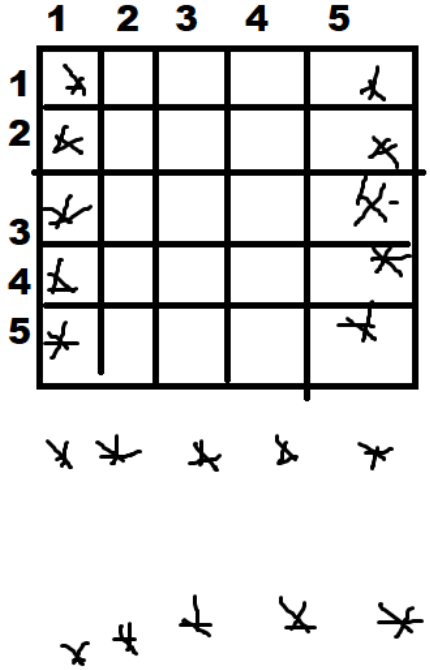
```
for i in range(1,6):
  for j in range(1,6):
    if i==j:
      print("*",end=' ')
    else:
      print("  ",end=' ')
  print()
```





```
for i in range(1,6):
  for j in range(1,6):
    if i==j or i+j==6:
      print("*",end=' ')
    else:
      print("  ",end=' ')
  print()
```

```
for i in range(1,6):
  for j in range(5,0,-1):
    if i==j:
      print("*",end=' ')
    else:
      print('  ',end=' ')
  print()
```



```
for i in range(1,6):
  for j in range(1,6):
    if j==1 or j==5:
      print("*",end=' ')
    else:
      print("  ",end=' ')
  print()
```

```
for i in range(1,6):
  for j in range(1,6):
    if i==1 or i==5:
      print("*",end=' ')
    else:
      print("  ",end=' ')
  print()
```

Example	Output
space=4 for i in range(1,6): for s in range(space): print(" ",end=' ') for j in range(1,i+1): print("* ",end=' ') print() space=space-1	* *

```
space=0
for i in range(5,0,-1):
   for s in range(space):
      print(" ",end=' ')
   for j in range(1,i+1):
      print("* ",end=' ')
   print()
   space=space+1
```

```
      *  *
        *
```

Nested While Loop

Defining while loop inside while loop is called nested while.

Syntax:

```
while <condition>:
  statement-1
  statement-2
  while <condition>:
      statement-1
      statement-2
```

In while, block of statements are executed until given condition.

Example

Write a program to generate all armstrong numbers between 100 to 999

```
num=100
while num<=999:
   num1=num
   s=0
   while num1>0:
      r=num1%10
      s=s+(r**3)
      num1=num1//10
   if s==num:
      print(num)
   num=num+1
```

Output

```
153
370
371
407
```

break, continue

These statements are called branching statements.

break

"break" is a keyword, which represent branching statement. Break statement is used inside looping statements (while or for). This statement is used to terminate execution of while or for loop in between.

Example:

```
for i in range(1,101):
   print("Hello")
   break

n=1
while  n<=10:
   print("Bye")
   n=n+1
   break
```

Output

Hello
Bye

Example:

```
# Login Application

valid=False
for i in range(3):
   uname=input("UserName :")
   pwd=input("Password :")
   if uname=="nit" and pwd=="n123":
      valid=True
      break
   else:
      print("invalid username or password")

if valid:
   print("Welcome ")
else:
   print("3 attempts are completed, your account is blocked...")
```

Output
UserName :nit
Password :n123
Welcome

UserName :nit
Password :xyz
invalid username or password
UserName :abc
Password :xyz
invalid username or password
UserName :aaa
Password :bbb
invalid username or password
3 attempts are completed, your account is blocked...

Example:
```
# Write a program to find input number is prime

num=int(input("Enter any number "))
i=1
c=0
while i<=num:
    if num%i==0:
        c=c+1
    if c>2:
        break
    i=i+1

if c==2:
    print("Prime")
else:
    print("Not Prime")
```

Output
Enter any number 4
Not Prime

Enter any number 5
Prime

continue

"continue" is a keyword which represents branching statement
This statement is used inside while or for loop.
This statement is used to move the execution to the beginning of the loop (while or for loop).

Example:

```
for num in range(1,21):
    if num%2==0:
        continue
    print(num,end=' ')


print()
for i in range(1,6):
    print("Hello")
    continue
    print("Bye")
```

Output

```
1 3 5 7 9 11 13 15 17 19
Hello
Hello
Hello
Hello
Hello
```

While ..else and for..else

Syntax:	**Syntax:**

while <condition>:
statement-1
statement-2
else:
statement-3
statement-4
	for variable in iterable:
	statement-1
	statement-2
	else:
	statement-3
	statement-4

Points

1. Else block is executed, after execution of while or for
2. Else block is not executed, if while or for loop is terminated unconditionally using break

Example:

```
# Write a program to find factorial of input number

num=int(input("Enter any number "))

fact=1
for i in range(1,num+1):
    fact=fact*i
else:
    print(f"Factorial of {num} is {fact}")

fact=1
i=1
while i<=num:
    fact=fact*i
    i=i+1
else:
    print(f"Factorial of {num} is {fact}")
```

Output

```
Enter any number 3
Factorial of 3 is 6
Factorial of 3 is 6
```

Example:	**Output**

for i in range(1,6):
  print("Hello")
  break
else:
  print("Bye")
 	
i=1
while i<=5:
  print("Hello")
  i=i+1
	Hello
	Hello

break else: print("Bye")	

Data structures or Collections or Iterables

What is collection in python?
Collection type represents more than one value or object.
Collections allows to group individual objects into one object.

1. Perform aggregate operations
2. By grouping data, we can transfer or send data from one place to another place
3. It allows referring data with one name (OR) does not required create multiple variables to store more than one value.