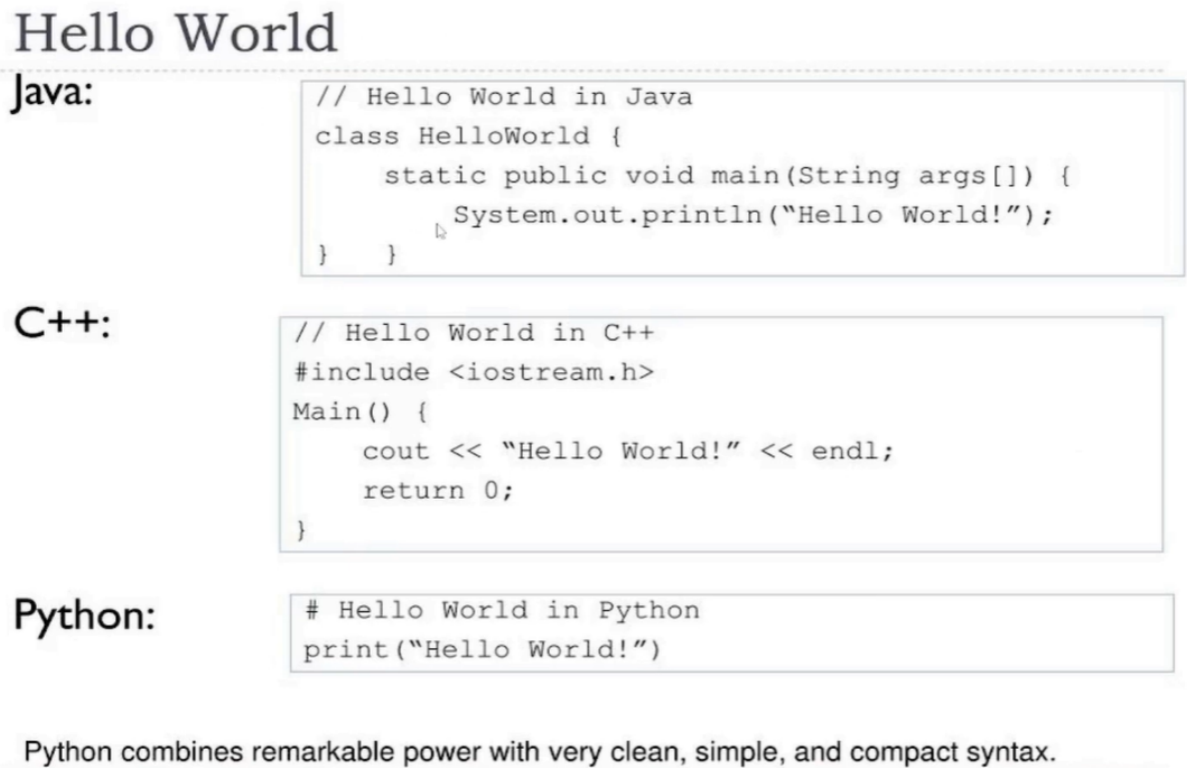
What is Python?

- Python is a high level programming language like C, C++
- Python is a simple, high level, interpreted, general purpose, dunamically typed and object oriented programming language created by Guido Van Rossum in 1991.

Python is simple

- Python is easy to use.
- python's programs are simple to write and hence it is easy to learn.
- Lots of built-in modules, packages and Frameworks



Python is high-level

- More user-friendly
- There is automatic memory management
- Rich set of libraries and functions ex. len()

Python is interpreted language:-

- Final execution of python program is done by interpreter.
- Interpreter scan the python program line by line.

Python is general purpose programming language:-

- Python is used for web development, machine learning, artificial intelligence, data analysis, data science, web scraping, scripting, scientific computing, software dev etc

Python is dynamically typed

- Like C or C++, we don't need to specify datatypes of identifiers.
- Python evaluates datatypes at runtime

Python is object oriented programming language

- In python, we can create classes and objects.
- Python supports nearly all concepts of object oriented style

---

Features of Python

- 1. Simple syntax and easy to learn
- 2. General purpose programming language
- 3. Dynamically typed programming language
- 4. Interpreted & dynamically typed language.
- 5. Python is free and open source language
- 6. Large standard libraries

Python supports automatic memory management.

- 1. Memory allocation :- Python memory manager
- 2. Memory de - allocation :- Garbage collector

Python is cross platform

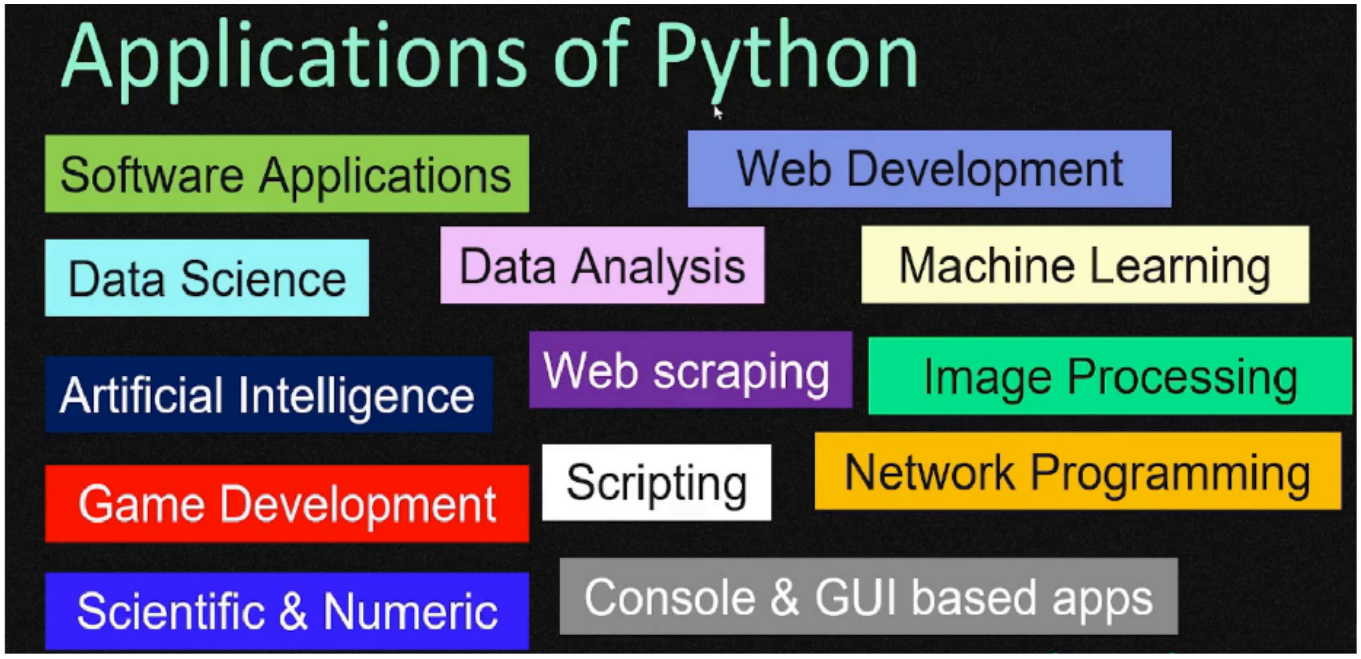
- python is platform independent.
- We can run python programs on any platform like Windows and Linux etc. It is portable language.

Python supports multiple programming paradigms.

- Python supports procedural, modular, functional, object oriented approaches.

Development and debugging cycle is very fast in python

- No need to care about datatypes
- Interpreted nature and clean code makes debugging faster
- python also provides some debugging features



Web applications :- Django, flask, pyramid Frameworks

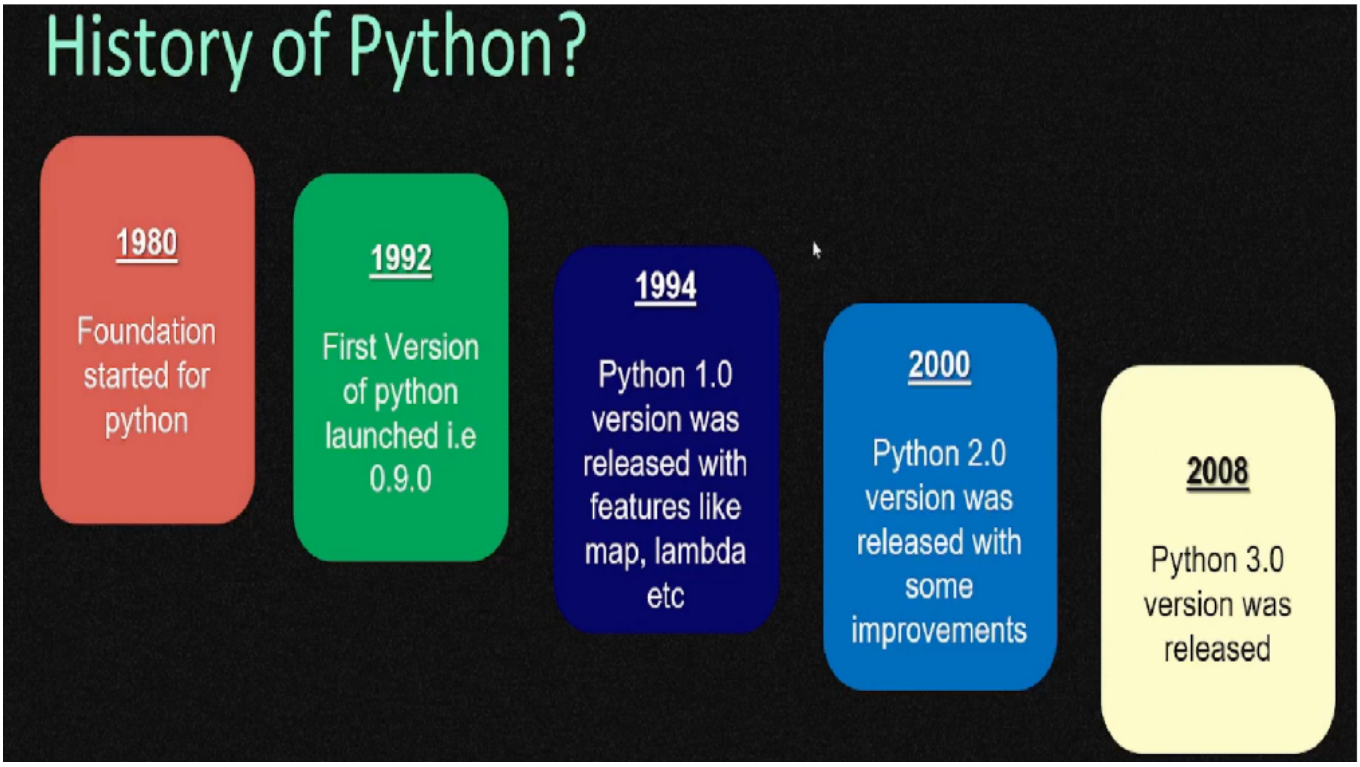
GUI application:- libraries like Kivy, Tkinter

Scientific and numeric applications:- Scipy, pandas, numpy

Data science:- numpy, pandas,seaborn, matplotlib

Image processing:- OpenCv library

Game Development:- Pygame library



Python is influenced by following programming languages:-

1. ABC Lang.
2. Modula -3 Python software foundation (PSF), A non-profit organization holds the copyright of python

How to run Python Program

1. By using text editor and CMD/windows powershell
2. By using python command line window
3. By using IDLE
4. By using different IDE'S
5. Using Anaconda distribution & Jupyter

What is IDE?

- IDE stands for Integrated Development Environment provides tools and environments to create applications.
- IDE provides programming environment

Features:-

- Syntax highlighting
- Syntax checking
- Syntax completion
- Error checking
- Interactive Console
- Integration with Tools and Frameworks
- Plug -ins
- Integratation with Git and GitHub

Different IDEs for python developement.

1. IDLE

- Provided by python
- Only for learning purpose
- CPython compiler is used

2. Spyder

- specifically designed for scientific and data-related programming.
- part of anaconda distribution
- data analysis, visualization, and scientific computing.

3. Pycharm

- specially designed for python community.
- used by professionals.
- provides all tools and frameworks of python.

4. Thrid - party IDEs

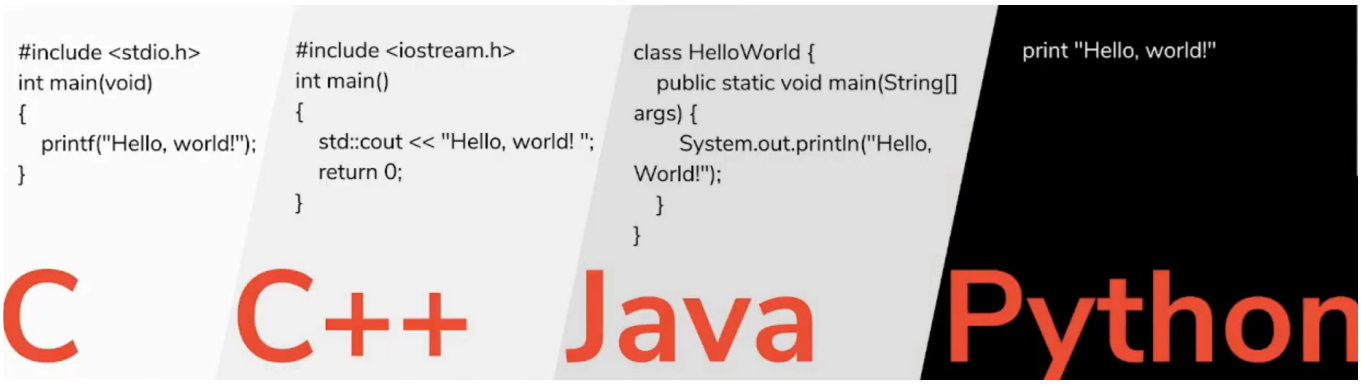
- VS Code
- Jupyter
- Atom

What is IDLE

- Integrated Development Learning Env
- It is an IDE provided by python.
- When you install python, IDLE application gets installed.

Two ways to run programs

- using interactive shell
- using a script file



Need of Variables / identifiers

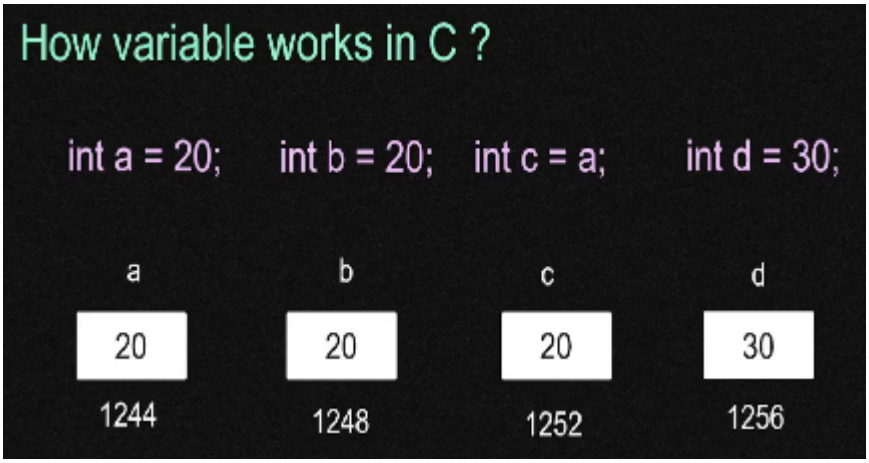
- For data manipulation
- For reusability of data
- For storing results of expressions
- Example: pi = 3.14159 r = 3 aoc = pi*rr* print(2*pir*)

> ## aoc variable result can be used anywhere in module. reuseability.

Python is a dynamic language. Type of the variable get resoved at runtime by python interpreter.

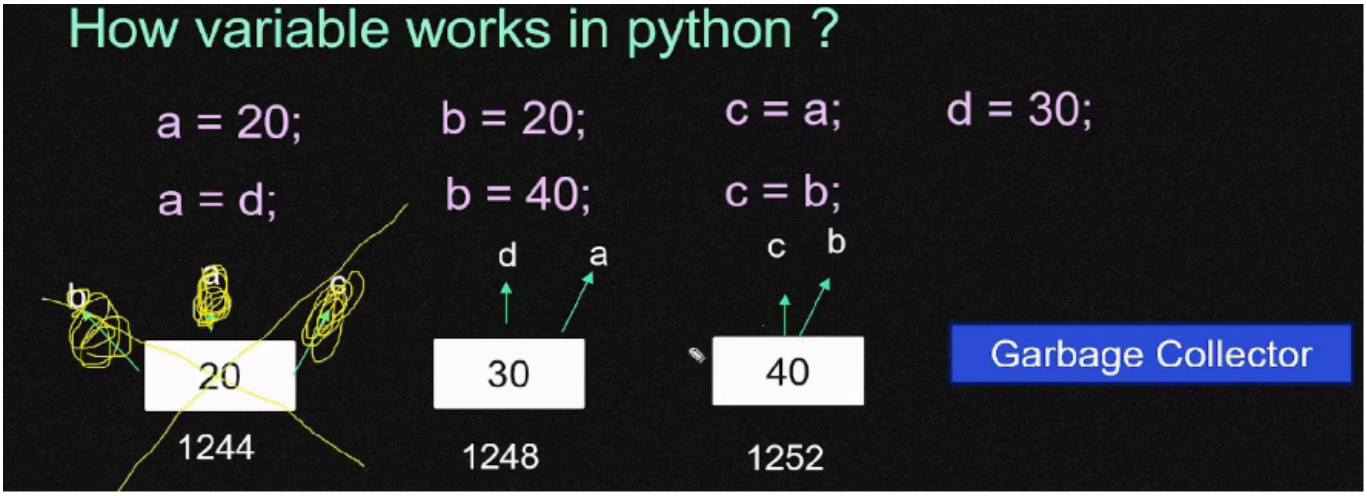
- How to create variables in python ? In C :- int age = 20; In python :- age = 20 Note :- No need to specify datatype while declaring

How variable works in python ?



> In python when we write a = 20, memory gets alloated for value 20 and a is refering to this value, hence in python we call variable a

> name as tag.So later when we declare b = 20, it does not occupy another memory space for 20, rather python creates another tag b and refers to same memory location.



Example code

```
a = 20
print("a:",id(a))

b = 20
print("b:",id(b))

c = a
print("c:",id(c))

d = 100
print("d:",id(d))
d = 200
print("d:",id(d))
```

we can declare variables like below

```
#creating multiple variables in one line
fullname = 'John Doe'
age = 56
salary = 34568.7
print(fullname,age,salary)

nm, ag, sal = "Hugh Jackman", 59, 7000000
print(nm,ag,sal)
#OR
```

```
print(nm),print(ag),print(sal)
```

```
#assign single value to multiple variables
num1 = 23
num2 = 23
num3 = 23

num1= num2 = num3 = 23
print(num1,num2,num3)
```

Rules to declare variables:

1. variable must start with alphabet or an underscore(_).
    - age, _age, __age, --> allowed | 23age -->not allowed
2. can't use special symbols except underscore (_).
    - my_age --> allowed | my$age --> not allowed
3. can't use keywords as variable names.
    - Tue, def, import are reserved keywords
4. Variable name should not start with digit
    - 23_age --> not allowed
5. No whitespaces allowed in variable names.
    - my age --> not allowed
6. variables in python are case-sensitive
    - Name = "Hugh" | name = 'John'

DO's and Don't

1. Always create meaningful variable names.
2. Recommended to use lowercase letters.
3. Separate multiple words by underscore.
4. Avoid funny and meaningless variable names.
5. class name should start with uppercase letters

```
# How to delete a variable
a = 100
b = "hello"
print(a,b)

del a,b
print(a,b) # NameError: name 'a' is not defined
```

```
#variable annotations
name: str = "John"
print(name)

name = 23
print(name)
```

> c/java allocates memory for variables.
>
> python allocates memory for values and not for variables.
>
> Python has efficient memory management.

Keywords in python

- Keywords are special words in any programming language.
- These are reserved words in python
- Every keyword has specific meaning

```
import keyword

print(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']
```

Python code gets Interpreted OR Compiled?

> Python Code --> Compiler --> Interpreter --> Computer / CPU

> Write Code inEditor --> file.py --> Compiler (checks for syntax etx)-- --> Btecode (optional .pyc) --> PVM (Python Virtual Machine) --> Interpreter --> CPU

Why we need both ? Compiler & Interpreter

> Portability...

No platform dependency : write code on one platform and run on other platforms without any change.

> Dynamic Typing..
>
> This operation requires some additional checks. These checks are performed at compilation time
>
> Security .-
>
> bytecodes are not human understandable.
>
> Optimizations :-
>
> The Python apply certain optimizations during bytecode generation peephole optimization :- Process to execute bytecodes more efficiently.
>
> Pre-compilation for Packaging
>
> Packages and modules are always pre-compiled

What is Bytecode and .pyc file ?

- bytecode is a low-level representation of your Python source code
- It is generated by Compiler
- Bytecode is not directly readable by humans
- Computer also cannot understand bytecodes
- .pyc file is an itermediate file which get generated after compilation.
- this stored puthon byte code and we can easly inport this file.

Python Module Import and Caching

When you create modules such as 'main.py' and 'example.py', and 'example.py' imports main, Python automatically generates a special directory: • Cache Directory: A folder named 'pychache' is created. • Cached Files: Inside it, Python stores a compiled bytecode version of 'main.py' (for example:- main.cpython-311.pyc). • Execution Optimization: • When any module imports main, Python uses this cached .pyc file instead of re-reading and re-parsing main.py. • This speeds up execution across the entire application. • Automatic Updates: If you modify 'main.py', Python will regenerate the cached file to keep it in sync. • Safe to Delete: If you remove the cache folder, Python will recreate it the next time the module is imported.

Key Notes • The cache improves performance but does not replace the source .py file. • Cached files are interpreter-specific (e.g., CPython 3.11). • Developers usually don’t need to manage these files manually

---

> Why Python uses Interpreter ? There are some advantages
>
> 1. Dynamic typing
> 2. Optimizations :- "peephole optimization" to make bytecode execution more efficient.
> 3. Efficient Exception Handling

> ## Why Python is always referred as In!erpreted Language .
>
> Final Execution of program is done by Interpreter. You don't need to compile the code everytime

> Does python creates .exe file ?
>
> Python itself does not create standalone .exe files for your Python scripts or programs internally. The Python inter*preter reads and executes the script's code line by line.

functions in python

> Syntax : def function_name(): function statements

To call function

> function_name()

```
num = 5
print(num* num)

num = 6
print(num* num)
# for multiple statements writtting again and again, we go for functions

# user defined functions:
def square(num):
  print(num* num)

square(2)
square(5)

# built-in function
print(type(10))
```

```
# built -in function
# in CPython id() return memory address
x = 10
print(id(x))

my_name = "John Doe"
print(id(my_name))

# bin() :  returns binary equivalent to integer passed.
# bin(integer) : it only works with interger values
print(bin(10)) # output: 0b1010 , binary representation of value 10
print(bin("abc")) # TypeError: 'str' object cannot be interpreted as an integer
```

```
#--------------
# pow() function:-
# syntax:- pow(x,y,z):-
# x:- base number
# y:- exponent
# z:- modulous (Optional)

print(2**3) # exponential operator : 8

print(pow(2,3)) # inbuilt function : 8

print(2.0**3) # exponential operator : 8.0

print(pow(2.0,3)) # inbuilt function : 8.0

# by passing 3rd args it calculated modulous:
# 2**3 = 8, 8 % 3 = 2 --> ouput - 2
print(pow(2,3,3)) # inbuilt function : 2

# -----------

# round() in python:-
# returns rounded version of specified number
# syntax:- round(number,digits)
# where number:- number specified for rounding
#   digits:- number of decimals to use
#   (optional argument and default is e)

num = 123.4567
print(round(num,2)) # 123.45
print(round(num,1)) # 123.4
print(round(num)) # 123 : default value is 0, zero

# -----------
# abs() in python:-
# - returns -absolute number
# syntax:- abs (value)

print(abs(-10))
print(abs(-11.22))
print(abs(0))
print(abs(11.33))
```

```
# math module: -basic mathematical operations and constants such as trigonometric
functions,logarithmic functions
#ex.
import math
print (math. sqrt(25)) # Square root
```

```
print(math.sin(math.pi / 2)) # Sine

# cmath module:- The cmath module is used for complex number arithmetic.
import cmath
z = complex(2, 3)
print(cmath.sqrt(z)) # Complex square root

# statistics module:-
import sympy
x = sympy.symbols('x')
equation = x**2 - 4
solutions = sympy. solve(equation,x)
print (solutions)

# sympy:- SymPy is a Python library for symbolic mathematics.
# ex.solving equations and expressions.

import sympy
x = sympy.symbols('x')
equation = x**2 - 4
solutions = sympy. solve(equation,x)
print (solutions)
```

To check how many built in functions are there in python use code:

```
import builtins
print(dir(builtins))
```

Below are datatypes available in python:-

- Numeric types:- int, float, complex (special type)
- Sequence types:- str, list, tuple
- Mapping types:- dict
- Set types:- set, frozenset
- Boolean Type:- Bool
- Binary Types:- bytes, bytearray
- None type: — None Type

```
# built in type() and isinstance() functions demo:

def square():
  num =int(input("Enter a number: "))
  if(isinstance(num,int)):
    return "Square of {} is: {}".format(num,num * num)
  else:
    inputtype = type(num)
    return "Input can not be {}. please provide int value".format(inputtype)

print(square())
```

Datatypes in Python:

1. int:

- Numeric types
- contains intergers
- can be of any length
- ex. ..-3,-2,-1,0,1,2,3..
- ex. target = 180

2. float:

- Numeric type
- floating point numbers
- accurate upto 15 decimal places
- ex. 3.2,4.0,8.9,-12.4
- ex. temperature = 37.4
- Exponential data falls under 'float' class
- ex. Data = 2e/E7 --> 2* 10**7 --> output is stores as float value
- salary = 2e3 --> op: 2000.0 --> 2 * 10**3

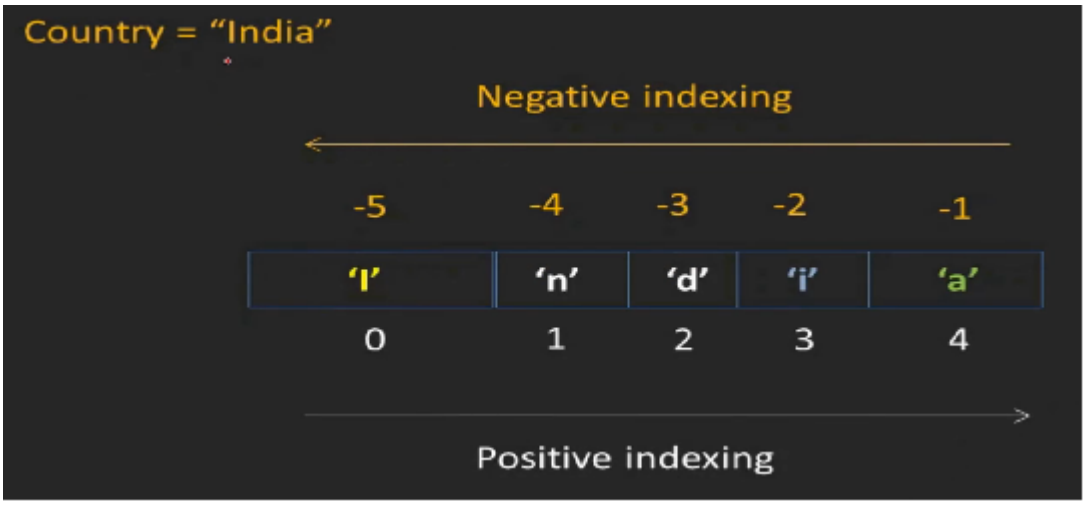
3. complex:

- written in form of a+bj
- 'a' is real part.can be integer or float.
- 'bj' is an imaginary part.can be integer or float.
- Ex. c = 5+4j

4. string:

- there is no char type in python
- single char will be stored as string of length 1
- Sequence of characters.
- Anything enclosed in quotes.

- triple quotes are used for multi-line strings.
- Ex. name = "hello"



- to declare string we can use '','"","'","""
- mostly ''' or """ quotes are used to write formatted output or multi-line comments
- ex.

```
print('''
address:
101, ABCD,
Pune,
411011
''')
str= 'hello'
print(len(str))

print(str[2:4])
```

5. Sequence - types

a. List:

- list of data's having different data types.
- Ordered mutable sequence of items.
- Items separated by commas enclosed in []
- +ve as well as -ve indexing is allowed.
- Ex. lst = [1,2,'apple',[23.33,True],2+3j,4,'hello','q']

b. Tuple

- Collection of different data types.
- Immutable,ordered
- can be accessed using indexing
- Items are separated with comma and enclosed in ().

- Ex. t = (22,4.5,"John",[1,2,3])

c. Dictionary

- Unordered set of key-value paires.
- Key has primitive datatype and value has any datatype.
- Key-value paires are separated by comma.
- Ex. current_rates = {EUR : 107 , THB : 2.87, USD : 90.6 }

d. set:

- Unordered collection of unique items. No duplicate values are allowed. They gets ignored.
- Items are written inside { } and separated by commas.
- Items are not in order.
- hence can not be accessed using indexing
- Ex. a = {1,2,3,4,'apple', 23.33}

e. range:

- It gives immutable sequence of numbers betn start and stop.
- Syntax:- range(start,stop,step)
- Default starting is O and step is 1.
- Ex. range(0,11) --> 0,1,2..10
- Ex. range(0,11,2) --> [0, 2, 4, 6, 8, 10]

```
print(list(range(0,11))) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(list(range(0,11,2))) # [0, 2, 4, 6, 8, 10]
print(list(range(0,11,3))) # [0, 3, 6, 9]
print(list(range(1,10,3))) # [1, 4, 7]
```

f. None

- None datatype means an object that doesn't contain any value.
- Ex. a = None

g. bool

- Two values, True and False.
- Used to determine given statements are true or false.
- True is non-zero and False is O.
- Ex. a = True

Operators in Python

---

1. Arithmetic Operators

Used for basic mathematical operations.

```
# Arithmetic Operators Demo
a = 10
b = 3

print("Addition:", a + b)       # 13
print("Subtraction:", a - b)    # 7
print("Multiplication:", a * b) # 30
print("Division:", a / b)       # 3.333...
print("Floor Division:", a // b)# 3
print("Modulus:", a % b)        # 1
print("Exponent:", a ** b)      # 1000
```

---

2. Comparison Operators

Used to compare values; return `True` or `False`.

```
# Comparison Operators Demo
x = 5
y = 10

print("Equal:", x == y)         # False
print("Not Equal:", x != y)     # True
print("Greater:", x > y)        # False
print("Less:", x < y)           # True
print("Greater or Equal:", x >= 5) # True
print("Less or Equal:", y <= 10)   # True
```

---

3. Logical Operators

Combine conditional statements.

```
# Logical Operators Demo
p = True
q = False

print("AND:", p and q)   # False
print("OR:", p or q)     # True
print("NOT:", not p)     # False
```

---

4. Assignment Operators

Used to assign values and update variables.

```
# Assignment Operators Demo
num = 10
print("Initial:", num)

num += 5   # num = num + 5
print("After += :", num)

num -= 3   # num = num - 3
print("After -= :", num)

num *= 2   # num = num * 2
print("After *= :", num)

num /= 4   # num = num / 4
print("After /= :", num)

num %= 3   # num = num % 3
print("After %= :", num)
```

---

5. Bitwise Operators

Operate on binary representations.

```
# Bitwise Operators Demo
a = 6   # 110 in binary
b = 3   # 011 in binary

print("AND:", a & b)   # 2 (010)
print("OR:", a | b)    # 7 (111)
print("XOR:", a ^ b)   # 5 (101)
print("NOT:", ~a)      # -7 (two's complement)
print("Left Shift:", a << 1) # 12 (1100)
print("Right Shift:", a >> 1)# 3 (011)
```

---

6. Identity Operators

Check if two objects share the same memory location.

```
# Identity Operators Demo
list1 = [1, 2, 3]
list2 = [1, 2, 3]
list3 = list1
```

```
print("list1 is list2:", list1 is list2)   # False
print("list1 is list3:", list1 is list3)   # True
print("list1 is not list2:", list1 is not list2) # True
```

---

7. Membership Operators

Check if a value exists in a sequence.

```
# Membership Operators Demo
nums = [1, 2, 3, 4, 5]

print("3 in nums:", 3 in nums)       # True
print("10 not in nums:", 10 not in nums) # True
```

☑ Quick Recap

- **Arithmetic** → Math operations
- **Comparison** → True/False checks
- **Logical** → Combine conditions
- **Assignment** → Update values
- **Bitwise** → Binary-level operations
- **Identity** → Memory reference check
- **Membership** → Sequence membership

Questions

Q1. Which operator is used for floor division in Python?

- A) `//`
- B) `/`
- C) `**`
- D) `%`

  Answer: A) `//`
  Hint: It discards the fractional part of division.
  Explanation: The `//` operator performs floor division, returning the integer quotient without the remainder.

Q2. What will be the output of `2 ** 3`?

- A) 6
- B) 16

- C) 8
- D) 9

  Answer: C) 8

  Hint: It raises the first number to the power of the second.

  Explanation: `2 ** 3` means 2 raised to the power of 3, which equals 8.

Q3. Which operator checks if two variables point to the same object in memory?

- A) `is`
- B) `!=`
- C) `==`
- D) `in`

  Answer: A) `is`

  Hint: It compares identity, not equality.

  Explanation: The `is` operator checks if two variables refer to the same object in memory.

Q4. What is the result of `10 % 3`?

- A) 3
- B) 7
- C) 1
- D) 0

  Answer: C) 1

  Hint: It gives the remainder after division.

  Explanation: 10 divided by 3 leaves a remainder of 1, so `10 % 3 = 1`.

Q5. Which operator would you use to check if `5` exists in a list?

- A) `==`
- B) `&`
- C) `in`
- D) `is`

  Answer: C) `in`

  Hint: It checks membership in sequences.

  Explanation: The `in` operator is used to check if an element exists in a list, tuple, or string.

Exercise 1: Arithmetic

Write a program to calculate the area and perimeter of a rectangle with length = 12 and width = 5 using arithmetic operators.

Exercise 2: Comparison

Check if a given number is divisible by both 3 and 5. Print `True` if it is, otherwise `False`.

Coding Challenges: Predict the Output

Challenge 1: Arithmetic

```
x = 7
y = 2
print(x // y)
print(x % y)
print(x ** y)
```

Question: What will be printed?

Challenge 2: Comparison

```
a = 15
b = 20
print(a > b)
print(a <= 15)
print(a == b - 5)
```

Question: Predict the boolean results.

Challenge 3: Logical

```
p = True
q = False
r = True

print(p and q)
print(p or q)
print(not r)
```

Question: What values will appear?

Challenge 4: Assignment

```
num = 4
num *= 3
num -= 5
num += 2
print(num)
```

Question: What is the final value of `num`?

Challenge 5: Bitwise

```
a = 5   # 101 in binary
b = 3   # 011 in binary

print(a & b)
print(a | b)
print(a ^ b)
print(a << 1)
print(b >> 1)
```

Question: Predict the integer results.

Challenge 6: Identity

```
list1 = [10, 20]
list2 = [10, 20]
list3 = list1

print(list1 is list2)
print(list1 is list3)
```

Question: Which comparisons return `True`?

Challenge 7: Membership

```
letters = ['a', 'b', 'c', 'd']
print('c' in letters)
print('z' not in letters)
```

Question: What will be printed?

Answer Keys

▶ Challenge 1: Arithmetic

```
x // y = 3
x % y  = 1
```

```
x ** y = 49
```

▶ Challenge 2: Comparison

```
a > b       → False
a <= 15     → True
a == b - 5  → True
```

▶ Challenge 3: Logical

```
p and q → False
p or q  → True
not r   → False
```

▶ Challenge 4: Assignment

```
num = 4
num *= 3 → 12
num -= 5 → 7
num += 2 → 9
Final Output: 9
```

▶ Challenge 5: Bitwise

```
a & b   → 1   (001)
a | b   → 7   (111)
a ^ b   → 6   (110)
a << 1  → 10  (1010)
b >> 1  → 1   (001)
```

▶ Challenge 6: Identity

```
list1 is list2 → False
list1 is list3 → True
```

▶ Challenge 7: Membership

```

```

```
'c' in letters     → True
'z' not in letters → True
```

What is PEP8?

PEP8

- Python Interprise Proposal
- A document that provides guidelines and best practices on how to write Python code.
- It was written in 2001 by Guido van Rossum
- these guidelines are helpful to enhance the readability and consistency of programs.
- Ex.l Naming conventions of variables, modules & packages.
- Ex.2. Variable name for constant and Class-name.

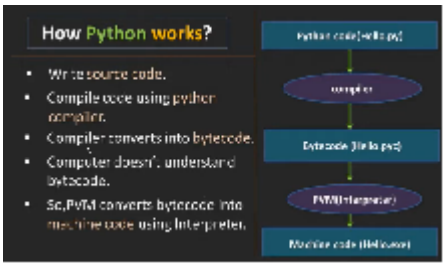
Q.1 Python case sensitive while dealing with identifiers? a) yes_ b) No c) Machine dependent d) None

---

Q.2 Which of the following is an invalid identifier? a) My_stringl b) while* #keyword c) foo d) __

Q.3 Why python is 'generaJ_purpose--programming' language? a) Python supports object-oriented concepts. b) Python is powerfull scripting language. c) Used in variety of application domains.* [ML, Scripting, Web Development, Data Science, AI etc.]

Q.4 Python is compiled or interpreted? a) Compiled -interpreted* b) interpreted c) Compiled d) None



Q.5 'IDLE' stand for....? Integrated Development Learning Environment It is an IDE given by python.org

---

Q.6 When we download python from it's official website,which compiler we get... ? a) jython b) Ccompiler* c) Ruby python d) Stackless python

---

Q.7 What is printed when following statement executed? day = "Thursday" day = 32.5 day = 19 print(day)

a) 32.5 c) Thursday b) 19* d) Error

Q.8 True or False: the following is a legal variable name in Python: A_good_grade_is_A+ a) True b) False* # + is not allowed as special char

---

Q.9 After the following statements, what are the values od x and y? x = 32 y = x x = 18 a) x is 18 and y is 22 c) x is 18 and y is 32* b) x is 32 and y is 32 d) x is 15 and y is 18

Q.10 What is printed when the following statement executed? x = 24 x = x - l print(x) a) 24 c) 22 b) 23* d) 25

Q.11 All keywords in python are in _ a) UPPER CASE B) LOWERCASE C) Capitalized d) None of above*

Q.12 Which of the following is true for variable names in Python? a) Underscore and ampersand are the only two special characters allowed. b) unlimited length* c) We can use whitespaces in variable name. d) Class name must start with uppercase letter.

Q.13 Which of the following is invalid statement? a) abc = 1,000,000 b) a_b_c = 1,000,000 c) a b c = 1000 2000 3000* d) a,b,c= 1000, 2000, 3000

Q.14 What is printed when the following statement executed? print(12,OOO)

a) 12,000 b) 12000 c) 12 000* # comma , changes into white space in print function, comma = whitespace d) Error : syntax-error

---

Q.15 Which of the following cannot be variable? a) **init** b) to C) in* d) on

---

Q.16 Who developed python language? a) Dennis Ritchie : C b) Bjarne Stroustrup : C++ C) Guido van rossum* :Python d) Brendan Eich :JavaScript

---

Q.17 what is the extension of python file? a) .txt b) .py* C) .pyc d) .c

---

Q.18 which symbol is used to write single line comment

a) # * b) // c) / d) $

Q.19 why python is called as 'dynamically typed language'?

- Python do not requires declaration of data types. When we create python object, python automatically interpretes it's datatype.

Q.20 write a python program to swap two numbers...

b=20 a,b=b,a print("value of a is:",a) print("value of b is:",b)

Modules in Python

Why Module?

- In any project, we have to write big programs.It become difficult to manage and organize programs
- So,we break down big program into small managable and organized files having same logic.These files are nothing but modules.
- it provides reusability to our code.

What are modules in python?:-

- A file containing python code is called as python module.
- Module contains functions definitions,class definitions, variables,runnable code,python statements you want to include in your python main file.
- If we try to write import statement multiple types - for same module; it ignores the rest declarations. if module is loaded for first time in memory it does not load it multiple times. DO remeber this.
- Also, we can not give module name as alredy provided buil-in module name. As it ignores multiple import calls for same module names, it first loads the built in module and ignores your custom module, so need to give different names.

```
# File - addition.py
def add(n1,n2):
  print(n1+n2)


def sub(n1,n2):
  print(n1-n2)

# ----------------------------------------

# File - calculator.py
from addition import add,sub
add(1,2)
sub(4,1)
#---------
# importing all
from addition import *
add(1,2)

sub(4,1)
#---------

from addition import add
add(1,2)
```

```
sub(4,1) #NameError: name 'sub' is not defined. Did you mean: 'sum'?


#-------------
import addition as a

a.add(3,5)

# ----------
import addition
addition.add(3,5)
```

Python has two types of modules:-

1. User defined modules:- modules created by us.
2. Built in modules:- modules given by python

```
# built in modules:

import math

n = math.factorial(5)
print(n)

# ----------

import math as m

print(m.pow(2,3))
```

---

dir()

dir() in python:- In Python, dir() is a built-in function that returns a list of all valid attributes (methods, variablels, and other attributes) for a given object.

syntax:- dir([object]) object:- value, function, class, module etc object is optional default value:- list of names in the current local scope.

```
# File 1 :  example.py

temp = 37.67

def display():
```

```
  '''  sample code  12345'''
  pass

def abcd():
  '''  sample code  abcdefg'''
  pass

# --------------

# File 2 : exm.py

x = 100
print(dir(x))

my_str = "sometext"
print(type(my_str))

print(dir(my_str))

'''
1. special methods : __methods__
2. special attributes : __attributes__
3. normal methods and normal attributes
'''

print(my_str.upper())
print(my_str.__class__)

# --------------------

import example as ex

print(dir(ex))
print(ex.display.__doc__)

# ----------
import main
print(dir(main))
print ( _ file )
print(dir(str))

# ----------------
import main
print (dir())

#module level scope
#functional level
import main
def demo():
  name "Hugh"
  print(dir())
demo ()
# ----------
```

```
### If we want to reload the module then use import importlib

#File 1 : example.py
print("Hello World")

# File 2 :  exm.py
import example
import example
import example

# this will print "Hello World" from example module only once though we have
imported it 3 times. As module gets loaded only once in memory.

# But if we need to still reload example module later somewhere in our code them=n
we need to import another helper module as below:

import importlib
import example
# import example
# import example

importlib.reload(example)
importlib.reload(example)

# This will show all modules available on your machine / system.
print(help("modules"))
```

all built in attribute in Python

- used in module programming
- What is all variable ? > from module_name import *
- This * is a wildcard, helps us to import everything from that module
- But, I don't want everything gets imported because of some reasons..

```
# File 1: example.py
x = 100

def personal_info():
  print("Hello, THis is Hugh Jackman")

def private_info():
  print("This is private info")

def display():
  print ("Hello")

# When we export all, all these modules gets loaded in imported module namespace.
Sometimes it may overwrite some functionalities like x value here is 100 andother
module is 2345. This is a drawback.
```

```
# To rstrict this importing with * wild card, we use __all__ attribute and
mentiones which modules we will be allowing in another namespace
__all__ =['x','personal_info']

# File 2: exm.py

# import example

from example import *

# When we call this dir() with import example, it does not show example modules
here. it only referes current namespace level modules.
# When we use from example import * we can see modules from other namespace
x = 2345
print(dir())

print(x)
personal_info()
# print(example)

# example.personal_info()
```

Advantages:-

- Prevent Accidental Import of Private Symbols
- Explicitly Define Public API
- Avoid Namespace Pollution
- Documentation and Readability

What is type casting?:

- Converting the value of one data type to another data type is called 'type casting' or 'type conversion'.
- Ex. 23.45 --> 23
- Twp types of conversion
    - Implicit type conversion
    - Explicit type conversion
- Implicit type conversion:- In Implicit type conversion,python automatically cönverts one data type to another data type. Generally,python promotes conversion of lower datatype to higher higher data type to avoid data loss.
- Explicit type conversion:- Programmer converts one data type to another data type. We have predefined functions for explicit type conversion. • int(value) • float(value) • str(value) • complex(value) • list(value) • tuple(value)

```
#implicit type conversion
x = 20 # int
y = 1.23 # float
x = x + y # float
print(z) # 21.23 : float
# int, float, str

#explicit type conversion
x = 20 # int
y = '1.23' # string
x = x + y # TypeError: unsupported operands
print(z) # 21.23 : float
# Type-casting
# Primitive data types : int, float, complex
# Sequencial data types: str, list, tuple

value = 12.5
print(value)
value1 = int(value)
print(valuel)

value = 12
print(type(value))
valuel = float(value)
print (type(valuel))
```