

1. Explain the key features of Python that make it a popular choice for programming?

Ans. Python is one of the most popular and beneficial programming language which is accessible for beginners and efficient for experienced developers. Python focuses on data industry and it is easy to adapt which makes it more interesting to read and write the code. It is simple like English language. Python has a lot of libraries (Python libraries are collection of pre-written code that programmers can use to perform tasks.) Python helps new developers to master it quickly and makes code easier to grasp. Python is easily accessible on various operating system like Windows, MacOS etc. Python popularly used in Web Development, Data science and machine learning, Software development. Python has the biggest contribution in the technical field and it is highly demandable in market that makes it more popular choice among beginners and a powerful tool for experts across different industries like technology, finance, healthcare, and education. Python has strong community support which make it an ideal choice for a wide range of programming applications.

2. Describe the role of predefined keywords in Python and provide examples of how they are used in a Program?

Ans. Python predefined keywords also known as reserved words which have sepecific meaning and purpose. It cannot be use as identifier (like variables). These predefined keywords define the whole structure, control flow, data handling, and error management. Using keywords appropriately is key to writing clean and effective Python code.

There is a List of Python Keywords python

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

**Here some examples of predefined keywords *type help ('keywords') to find the keywords.*

```
print ("Hello Aayushi")
```

```
➞ Hello Aayushi
```

Double-click (or enter) to edit

```
a = 5 #this is variable that is also known as identifier, it is reserved memory space to store value.
```

```
a
```

```
➞ 5
```

```
b = 9.6 #Value with decimal is the float value
```

```
b
```

```
➞ 9.6
```

```
c = "Aayushi" #word written in English language is known as str in python
```

```
c
```

```
➞ 'Aayushi'
```

```
type(c)
```

```
➞ str
```

```
type(b)
```

```
➞ float
```

```
type(a)
```

```
➞ int
```

```
e = True
```

```
type(e)
```

```
bool
```

```
f= False
```

```
type(f)
```

```
bool
```

True - False #True and False are instances of boolean. True is equivalent to integer 1 and False is equivalent to the integer 0.

```
1
```

```
False - True
```

```
-1
```

```
e = None
```

```
e
```

```
type (e)
```

```
NoneType
```

```
complex = x+iy
```

```
com = 5 + 6j
```

```
com = 5 + 6j
```

```
type (com)
```

```
complex
```

```
help ('keywords')
```

Here is a list of the Python keywords. Enter any keyword to get more help.

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

```
type ('return')
```

```
str
```

Indentation = the use of spaces or tabs at the beginning of a line of code to indicate a block of code

```
if 9 > 5: print("greater")
```

a = 3 #statement + fundamental block of code Name = Aayushi

statement can be of many types = expressions, assignment, loop statement, condition statement

Double-click (or enter) to edit

```
#input output, input takes every value as str
```

```
name = input("please enter your name")
```

```
➦ please enter your name5
```

```
name
```

```
➦ '5'
```

3. Compare and contrast mutable and immutable objects in Python with examples.

Ans. In python there are two types of objects.

MUTABLE OBJECT = Value that can be change after creating it is known as mutable object or container. Example - Lists, dictionaries, sets etc. when we modify the object it maintain its original source in memory. (means there is no new container is created.) supports item assignment.

```
#Mutable object
```

```
list_cont = [1,2,3,4.5,5+7j,True,"Ram"]
list_cont
```

```
➦ [1, 2, 3, 4.5, (5+7j), True, 'Ram']
```

▼ Default title text

```
# @title Default title text
#In python indexing starts from 0
#There are two types of indexing
#Positive indexing = starts right to left from 0
#Negative indexing = starts left to right from -1
```

```
#by above example of mutable object we can easily check the indexing.
```

```
list_cont[4]
```

```
➦ (5+7j)
```

```
#Modification
```

```
list_cont[5] = "False"
list_cont
```

```
➦ [1, 2, 3, 4.5, (5+7j), 'False', 'Ram']
```

list_cont is mutable,we can add elements to it. This change affects the original list without creating a new object. In modification we have changed "True" with "False" without creating a new object.

Immutable object = Value cannot be changed after creating it. It does not support item assignment. Modification to an immutable object will create a new object. Examples = String,integers,float etc. It is also thread safe because their statement cannot be changed.

```
#Immutable objects
```

```
str = "Hello"
print(str)
```

```
➦ Hello
```

```
#Modification
```

```
str = "Hello" + "Delhi"
print (str)
```

```
➦ HelloDelhi
```

Strings are immutable, so adding Delhi to string results in a new string object. The original string remains unchanged.

4. Discuss the different types of operators in Python and provide examples of how they are used?

Ans. Operators are the type of special or a specific keywords that are use to perform operations on variables or value. (Manage and to do computation and make decision using a particular data.)

1. **Arithmetic Operator** = Use to perform mathematical operations like addition, subtraction, Multiplication, division etc.

#Arithmetic Example

```
a = 4
b = 6
a+b
```

→ 10

```
x = 2
y = 8
x*y
```

→ 16

2. **Modulus Operator** = Denoted by % symbol to find the remainder of a division between two numbers.

#Modulus operator

```
74%9
```

→ 2

3. **Floor Operator** = Denoted by double slash (//) It divides two no. and rounds the result down to the nearest whole no. or integer.

#Floor Operator

```
#18//4
```

→ 4

4. **Comparison Operator** = It compares two value and give the boolean value of True or False. Example = ==, !=, >, <, >=, <=

#Comparison Operator

```
2!=9
```

→ True

#Need to use double = symbol to compare the value

```
8==10
```

→ False

```
43>93
```

→ False

5. **Logical Operator** = To combine logical statement. It carry out the logical operations and give the boolean value based on the result.

Example = and, or, not

#Logical Operator

```
True or False
```

→ True

```
False and True
```

→ False

```
False and False
```

 False

True and True


 True

True is not False

 True

6. **Not operator** = Inverts the true value of an expression. When applied to a condition, it returns True if the condition is False and False if the condition is True.

```
#Not operator
not True
```

 False

```
not False
```

 True


7. **Membership Operator** = use to check if a value is present in the sequence. Example = in, not in

```
#membership operator
```

```
a = "Uttarakhand"
"n" in a
```

 True

```
b = "Data"
"t" not in b
```

 False


8. **Identity operator** = compare the memory location of two objects to determine the similarity. Example = is, is not

```
#Identity operator
```

```
a = 10
b = 4
a is not b
```

 True

```
a is b
```

 False


9. **Bitwise operator** = use to perform calculation on integer. It is faster than Arithmetic operations. Example = &, ^, ~, <<, >>

```
#bitwise And operator
```

```
10&10
```

 10


```
bin(10)
```

 '0b1010'


```
3|5
```

 7

bin(3)

 '0b11'

bin(5)

 '0b101'

#Bitwise NOT

~4

 -5

#Bitwise xor operator

5^7

 2


#shift value = << (left shift operator = add last values)

#>> (right shift operator = remove last values)


33<<8

 8448


bin(33)

 '0b100001'

bin(8)

 '0b1000'

bin (8448)

 '0b10000100000000'

5. Explain the concept of type casting in Python with examples?

Ans. Type casting is also known as type conversion, it is a process of changing the data type of a value or object in python. While computing with operator there can be mismatch between the data types.

Type casting in Python is essential for converting data types to meet the needs of specific operations or functions. Implicit casting occurs automatically, while explicit casting uses functions like `int()`, `float()`, and `str()` for manual conversions.

Whatever we want to convert, the data type into then we need to put that inetger or float infront of the variable.

#type casting

a = "2"

b = 8

int (a)+b


 10

#changed string to interger

a = "2"

print(type(a))

print(type(int(a)))

 <class 'str'>
<class 'int'>

#converting float to integer

```
a = 7.8
type(a)
```

→ float

```
#converting into integer
```

```
int(a)
```

→ 7

```
x = 9.2
type(x)
float_value = int(x)
type(float_value)
```

→ int

```
#converting integer to float
```

```
a = 6
type(a)
```

→ int

```
b = float(a)
b
```

→ 6.0

```
type(b)
```

→ float

```
#string to float
a= "3.8"
type(a)
```

→ str

```
b = float(a)
type(b)
```

→ float

There are two types of type casting.

1. Implicit type casting = Python automatically converts one data type to another without explicit instructions.

Example = if we type a = 7 then type(a), it will give the integer (python has the tendency to understand the data type of its own.)

2. Explicit = The programmer manually converts the data type using specific functions.

int() – Converts a value to an integer, float() – Converts a value to a float, str() – Converts a value to a string.

```
#explicit typecasting = convert data type using in built function.
```

```
a="2"
type(a)
```

→ str

```
#Boolean type casting
bool(0)
```

→ False

```
bool(1)
```

 True


6. How do conditional statements work in Python? Illustrate with examples?

Ans. Conditional statements helps programs to make decisions based on the conditions, These statements help control the flow of a program by performing different actions based on whether a condition is True or False.

Example = if statement, if...else Statement, if...elif...else Statement and Nested if Statements.

```
#if condition:
    # Code to execute if condition is True
```


```
a = 89
if a > 9:
    print("the number greater than 9.")
```

 the number greater than 9.


```
#if...else Statement
```

```
#if condition: Code to execute if condition is True
#else: Code to execute if condition is False
```

```
weather = "sunny"
if weather == "sunny":
    print("I will not play outside")
else:
    print("I will go out")
```


 I will not play outside

```
age = 14
if age == ("18"):
    print("eligible for vote")
else:
    print("not eligible for vote")
```

 not eligible for vote


```
#if...elif...else Statement
```

```
a = 20
if a > 100:
    print("this block will executed if a is greater than 100")
elif a < 100:
    print("this block will excecuted if a is less than 100")
else:
    print("the number is equals to 100")
```

 this block will excecuted if a is less than 100

#Nested if Statements = we can nest if statements within other if statements to create more complex conditions.

```
a = 1
b = 6
if a > 5:
    if b > 5:
        print(" Both a and b is greater than 6")
    else:
        print("a is greater than 6 but b is less than 6")
else:
    print("a is not greater than 6")
```

 a is not greater than 6

```
sale = "Amazon"
age = 18
if sale:
```



```

if age >= 12:
    if age < 25:
        print("you are not eligible for sale")
    else:
        print("you are eligible for sale")
else:
    print("sale is not for people below 18 years")
else:
    print("sale is on Amazon")

```

you are not eligible for sale

7. Describe the different types of loops in Python and their use cases with examples.

Ans. In Python, loops are used to repeat a block of code multiple times, either a fixed number of times or until a specific condition is met. Python provides two main types of loops for executing a block of code multiple times: (while) loops and (for) loops.

While loop = while loop repeatedly executes a block of code as long as a given condition is True. It is best used when the number of iterations is not fixed and depends on a certain condition.

#while loop

```

a = 7
b = 1
while b<a:
    print(b)
    b = b+1

```

1
2
3
4
5
6

```

count = 4
while count > 0:
    print(count)
    count = count - 1

```

4
3
2
1

```

a = 7
b = 1
while b<a:
    print(b)
    b = b+1
else:
    print("This will be executed when the while loop will run")

```

1
2
3
4
5
6
This will be executed when the while loop will run

#Loop Control Statements
#break: Exits the loop entirely.

```

x = 9
y = 1
while y<x:
    print(y)
    y = y+1
    if y == 4:
        break

```

```
else:
    print("This will be executed when the while loop will run")
```

```
↩ 1
   2
   3
```

#continue: Skips the current iteration and moves to the next iteration.

```
x = 9
y = 1
while y<x:
    print(y)
    y = y+1
    if y == 4:
        continue
```

```
else:
    print("This will be executed when the while loop will run")
```

```
↩ 1
   2
   3
   4
   5
   6
   7
   8
   This will be executed when the while loop will run
```

For loop = The for loop is used to iterate over a sequence and execute a block of code once for each item in the sequence.

```
for a in "Uttarakhand":
    print(a)
```

```
↩ U
   t
   t
   r
   a
   k
   h
   a
   n
   d
```

```
a = "prakhar"
for h in a:
    print (h)
```

```
↩ p
   r
   a
   k
   h
   a
   r
```

#break statement (for) loop

```
for h in a:
    if h == "prakhar":
        break
    print(h)
```

```
else:
    print("this will executed when for loop ends without a break statement")
```

```
↩ p
   r
   a
   k
   h
   a
   r
```

this will executed when for loop ends without a break statement

```
#Using for Loop with range  
#syntax of range (start,stop,step)
```

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

```
↩ 0  
1  
2  
3  
4
```