INTRODUCTION TO PYTHON

* Identifiers :→

Identifiers is a user-defined name given to a variable, function, class, module, etc. The identifier is a combination of character digits and and underscore. They are case-sensitive i.e, 'num' and 'Num' and 'NUM' are three different identifiers in python. It is a good programming practice to give meaningful names to identifiers to make the code understandable.

* Rules for Naming Python Identifiers

- It cannot be reserved python keyword.
- It should not contain white space.
- It can be contain white space.
- It can be a combination of A-Z, a-z, 0-9 or underscore.
- It should start with an alphabet character or an underscore(_).
- It should not contain any special character other than an underscore(_).

* Examples of Python Identifiers

Valid identifiers:

- var1
- _var1
- _1_var
- var_1

Invalid identifiers:

- !var1
- 1var
- 1_var
- var#1
- var 1

Date / /
Page No.

* Keywords in Python

Python Keywords are some predefined and reserved words in Python that have special meanings. Keywords are used to define the syntax of the coding. The keyword cannot be used as an identifier, function, or variable name. All the keywords in Python are written in lowercase except True and False. There are 35 keywords in Python.

In python, there is an inbuilt keyword module that provides an iskeyword() function that can be used to check whether a given string is a valid keyword or not. Furthermore, we can check the name of the keywords in Python by using the kwlist attribute of the keyword module.

* Rules for keywords in Python
  - Python keywords cannot be used as identifiers
  - All the keywords in Python should be in lowercase except True and False

* List of python keywords

keywords	Description
and	This is a logical operator which returns true if both the operands are true else returns false.
or	This is also a logical operator which returns True if the operand is false else returns false.
not	This is again a logical operator it return

	Tirue if the operand is false else false.
if	This is used to make a conditional statement
elif	Elif is a condition statement used with an if statement. The elif statement is executed if the previous conditions were not true.
else	Else is used with if and elif conditional statements. The else block is executed if the given condition is not true.
for	This is used to create a loop.
while	This keyword is used to create a while loop.
break	This is used to terminate the loop.
as	This is used to create an alternative
def	It helps us to define functions.
lambda	It is used to define the anonymous function.
pass	This is a null statement which means it will do nothing.
return	It will return a value and exit the function.
Tirue	This is a boolean value.
False	This is also a boolen value

try	It makes a try - except statement.
with	The with keyword is used to simplify exception handling.
assert	This function is used for debugging purposes. Usually used to check the correctness of code.
class	It helps us to define a class.
continue	It continues to the next iteration of a loop
del	It deletes a reference to an object
except	Used with exceptions, what to do when an exception occurs
finally	Finally is used with exceptions, a block of code that will be executed no matter if there is an exception or not.
from	It is used import specific parts of any module
global	This declares a global variable.
import	This is used to import a module
in	It's used to check whether a value is present in a list, range, tuple, etc.

is	This are
none	This valu any com
nonlocal	It
raise	Th
yield	It ato
async	It
await	It the

* Sta Statement

- Cr
- Ex
- ru
- pr
- pa

statement do these an effect

is	This is used to check if the two variables are equal or not.
none	This is a special constant used to denote a null value or avoid. It is important to remember, 0, any empty containers (e.g. empty list) do not compute to None
nonlocal	It is declared a non-local variable.
raise	This raises an exception.
yield	It ends a function and returns a generator.
async	It is used to create asynchronous coroutine.
await	It releases the flow of control back to the event loop.

* Statements and Expressions

Statements ⇒ We need to write code that does something

a=2; b=1; print(letter);

statement separater

- Create a variable
- Create / assign a value to a variable
- run a separate piece of code elsewhere
- print something out to the screen
- pass some value on to another piece of code

Statements are the building blocks for our code that do these things. Any piece of code we write that has an effect is a statement.

Date / /
Page No.

An assignment statement will assign a value to a variable. set A = 3

A more complex assignment statement will evaluate an expression to assign a value to a variable.

set A = 3-2

Multiple statements combine to create more complex functionality

```
set A = 3-2
set B = 5
set C = A+B
```

Statements follow each other, and are carried out in order

Later, we'll look at complex statements that contain other statements and have their own functionality

- loop statements that repeat code or iterate (for, while)
- branching statements that allow code to make decisions (if, if-else)

The names of these complex statements (for, while, if, if-else) are often reserved. This means that they cannot be used for other purposes in our code - for example we usually cannot use 'for' as a variable name.

Expressions

Expression are pieces of code that will be evaluated to determine their value

- a mathematic operation
- that applies operators to the values stored in variables
- and provides a result

Expressions do not have effects, they merely

produce a va
ons, which m
can be used i

* Variables

Variables are
An Example o
ntational nam
object. Once a
can be referenc
terms, we ca
containers that
Rules for Pyth
- A python var
or the under
- A Python var
number.
- A Python va
meric charact
- Variable in Py
Name, and NAME
- The reserved wo
to name the

Example ⇒ #

produce a value. Statements can contain expressions, which means that the results of expressions can be used in our statements.

* Variables

Variables are containers for storing data values. An Example of a Variable in Python is a representational name that serves as a pointer to an object. Once an object is assigned to a variable, it can be referred to by that name. In layman's terms, we can say that variable in Python is containers that store values.

Rules for Python variables

- A python variable name must start with a letter or the underscore character.
- A Python variable name cannot start with a number.
- A Python variable name can only contain alphanumeric character and underscores (A-z, 0-9 and _)
- Variable in Python names are case-sensitive (name, Name, and NAME are three different variables)
- The reserved words (keywords) in Python cannot be used to name the variable in Python.

Example =>

```
# valid variable name
geeks=1
Geeks=2
Ge_e_ks=5
_geeks=6
geeks_=7
_GEEKS_=8

print(geeks, Geeks, Ge_e_ks)
print(_geeks, geeks_, _GEEKS_)
```

Output: 1 2 5
6 7 8

* Variables Assignment in Python

Here, we will define a variable in python. Here, clearly we have assigned a number, a floating point number, and a string to a variable such as age, salary and name. Code ⇒

```
# An integer assignment
age = 45

# A floating point
salary = 1456.8

# A string
name = "John"

print (age)
print (salary)
print (name)
```

Output: 45
1456.8
John

* Declaration and Initialization of Variables

Let's see how to declare a variable and how to define a variable and print the variable. Code ⇒

```
# declaring the var
Number = 100

# display
print (Number)
```

Output: 100

* Redeclaring variables in Python

We can re-declare the Python variable once we have declared the variable and define variable in python already.

Code ⇒

```
# declaring the var
Number = 100

# display
print ("Before declare : ", Number)

# re- declare the var
Number = 120.3

print ("After re-declare : ", Number)
```

Output : Before declare : 100
After re-declare : 120.3

* Python Assign Values to Multiple Variables
Also, Python allows assigning a single value to several variables simultaneously with "=" operators. For example ⇒

```
a = b = c = 10

print(a)
print(b)
print(c)
```

Output : 10
10
10

* Assigning different values to multiple variables
Python allows adding different values in a single line with "," operators. for example ⇒

```
a, b, c = 1, 20.2, "GeeksforGeeks"

print(a)
print(b)
print(c)
```

Output : 1
20.2
GeeksforGeeks

* Can We use the same Name for Different Types?
If we use the same name, the variable starts referring to a new value and type. for ex →

```
a=10
a="GeeksforGeeks"

print(a)
```

Output: GeeksforGeeks

How does + operator work with variables?
The Python plus operator + provides a convenient way to add a value if it is a number and concatenate if it is a string. If a variable is already created it assigns the new value back to the same variable for ex →

```
a=10
b=20
print(a+b)

a="Geeksfor"
b="Geeks"
print(a+b)
```

Output: 30
GeeksforGeeks

* Can we use + for different Datatypes also?
No use for different types would provide an error
Code →

```
a=10
b="Geeks"
print(a+b)
```

Output: TypeError: unsupported operand type(s) for +: 'int' and 'str'

* Global and Local Python Variables
Local variables in Python are the ones that are defined and declared inside a function. We can not call this variable outside the function. Eg →

```
# This function uses global variable s
def f():
    s = "Welcome geeks"
    print(s)

f()
```

Output: Welcome geeks

*Global variables in Python are the ones that are defined and declared outside a function and we need to use them inside a function. For ex →

```
# This function has a variable with
# name same as s
def f():
    print(s)

# Global scope
s = "I love Geeksforgeeks"
f()
```

Output: I love Geeksforgeeks

* Global keyword in Python

Python global is a keyword that allows a user to modify a variable outside of the current scope. It is used to create global variables from a non-global scope i.e. inside a fucn. Global keyword is used inside a function only when we want to do assignments or when we want to change a variable. Global is not needed for printing and accessing.

*Operators

In python programming, Operators in general are used to perform operations on values and variables. These are standard symbols used for logical and arithmetic operations. In this article, we will dark into different types of python operators.

• OPERATORS: These are the special symbols. Eg - +, *, /, etc.

*OPERAND: It is the value on which the operator is applied.

*Types of Operators :→

1) Arithmetic Operators
2) Comparison Operators
3) logical Operators
4) Bitwise Operators
5) Assignment Operators
6) Identity Operators and Membership Operators

1) Arithmetic Operators

Python Arithmetic operators are used to perform basic mathematical operations like addition, subtraction, multiplication and division.

Operator	Description	Syntax
+	Addition: adds two operands	x+y
-	subtraction: subtract two operands	x-y
*	Multiplication: multiplies two operands	x*y
/	Division (float): divides the first operand by the second	x/y
//	Division (floor): divides the first operand by the second	x//y
%	Modulus: returns the remainder when the first operand is divided by the second	x%y
**	Power: Returns first raised to power second	x**y

2) Comparis
In Python
compares
according

Opera
>
<
==
!=
>=
<=

3) Logical
Python L
AND, Log
rations.
conditional

2) Comparison Operators

In Python Comparison of Relational operators compares the values. It either returns True or False according to the condition.

Operator	Description	Syntax
>	Greater than: True if the left operand is greater than the right	x>y
<	less than: True if the left operand is less than the right	x<y
==	Equal to: True if both operands are equal	x==y
!=	Not equal to - True if operands are not equal	x!=y
>=	Greater than or equal to True if the left operand is greater than or equal to the right	
<=	less than or equal to True if the left operand is less than or equal to right	

3) Logical Operators

Python Logical operators perform Logical AND, Logical OR and Logical NOT operations. It is used to combine conditional statements.

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of operands is true	x or y
not	Logical NOT: True if the operand is false	not x

4) Bitwise Operators

Python Bitwise operators act on bits and perform bit-by-bit operations. These are used to operate on binary numbers.

Operator	Description	Syntax
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right Shift	x >>
<<	Bitwise left Shift	x <<

5) Assignment Operators

Date / /
Page No.

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of operands is true	x or y
not	Logical NOT: True if the operand is false	not x

4) Bitwise Operators

Python Bitwise operators act on bits and perform bit-by-bit operations. These are used to operate on binary numbers.

Operator	Description	Syntax
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x >>
<<	Bitwise left shift	x <<

5) Assignment Operators

Python Assignment operators are used to assign values to the variables

Date / /
Page No.

Operator	Description	Syntax
=	Assign the value of the right side of the expression to the left side operand	x = y+z
+=	Add AND: Add right-side operand with left side operand and then assign to left operand	a += b a = a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a -= b a = a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a *= b a = a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/= b a = a/b
%=	Modulus AND: Takes modulus using left and right operands and assign the result to left operand	a%= b a = a%b
//=	Divide (floor) AND: Divide left operand with right operand and then assign the value (floor) to left operand	a//= b a = a//b

Date / /
Page No.

** =	Exponent AND: Calculate exponent (raise power) value using operands and assign value to left operand	a**=b a=a**b
&=	Performs Bitwise AND on operands and assign value to left operand	a&=b a=a&b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b a=a|b
>>=	Performs Bitwise right shift on operands and assign value to left operand	a>>=b a=a>>b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a<<=b a=a<<b

6) Identity Operators

In Python, is and is not are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal do not imply that they are identical.

Example ⇒ The code uses identity operators to compare variable in Python. It checks if 'a' is not the same object as 'b' (which is true bcz they have different values) and if 'a' is the same object as 'c' (which is true because 'c' was assigned the value of 'a'). code ⇒

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

print(a is not b)
print(a is c)
```

Outp

* Member

In Python
operators
or variable

Example :→ T
'x' and 'y' i
value is
list, and
messages.
operators to

Output:

* Ternary O
In Python,
nal expressi
thing based
It was ad

Syntax : [o
[on

...ricise	a**=b
...l assign	a=a**b
, and	a&=b
	a=a&b
and	a^=b
	a=a^b
...perands	a>>=b
...d	a=a>>b
...rands	a<<=b
	a=a<<b

...ity operators
... are located
...o variables
... are identical
... compare
... not the
... they have
...bject as 'c'
...ed the value

Output: True
True

Date / /
Page No.

* Membership Operators in Python

In Python, in and not in are the membership operators that are used to test whether a value or variable is in a sequence.

Example :→ The code checks for the presence of values 'x' and 'y' in the list. It prints whether or not each value is present in the list. 'x' is not in the list, and 'y' is present, as indicated by the printed messages. The code uses the 'in' and 'not in' Python operators to perform these checks. code ⇒

```
x = 24
y = 20
list = [10, 20, 30, 40, 50]

if (x not in list):
    print ("x is NOT present in given list")
else:
    print ("x is present in given list");

if (y in list):
    print ("y is present in given list")
else:
    print ("y is NOT present in given list")
```

Output: x is NOT present in given list
y is present in given list

* Ternary Operators in Python

In Python, Ternary operators also known as conditional expressions are operators that evaluate something based on a condition being true or false. It was added to Python in version 2.5.

Syntax: [on_true] if [expression] else [on_false]

Example → The code assigns values to variables 'a' and 'b' (10 and 20, respectively). It then uses a conditional assignment to determine the smaller of the two values and assigns it to the variable 'min'. Finally, it prints the value of 'min', which is 10 in this case. code ⇒

```
a, b = 10, 20
min = a if a < b else b

print(min)
```

Output: 10

* Precedence and Associativity of Operators in Python

* Operator Precedence in Python

This is used in an expression with more than one operator with different precedence to determine which operation to perform first.

Example :→ The code first calculates and prints the value of the expression 10+20*30, which is 610. Then, it checks a condition based on the values of the 'name' and 'age' variables. Since the name is "Alex" and the condition is satisfied using the or operator, it prints "Hello! World." Code ⇒

```
expr = 10+20*30
print(expr)
name = "Alex"
age = 0

if name == "Alex" or name == "John" and age >= 2:
    print("Hello! World.")
else:
    print("Good Bye!!")
```

Outp

* Operator

If an e
with te
olativity
left to

Example
cal opera
of divisio
subtracti
The code
and their

Output:

* Data T
Python D
rization
value the
on a part
in Python
classes an
these cla
built-in

Output : 610
Hello! World.

Date / /
Page No.

* Operator Associativity in Python

If an expression contains two or more operators with the same precedence then Operator Associativity is used to determine. It can either be Left to Right or from Right to Left.

Example :- The code showcases various mathematical operations. It calculates and prints the results of division and multiplication, addition and subtraction, subtraction within parentheses, and exponentiation. The code illustrates different mathematical calculations and their outcomes. Code →

```
print (100 / 10 * 10)
print (5 - 2 + 3)
print (5 - (2 + 3))
print (2 ** 3 ** 2)
```

Output : 100.0
6
0
512

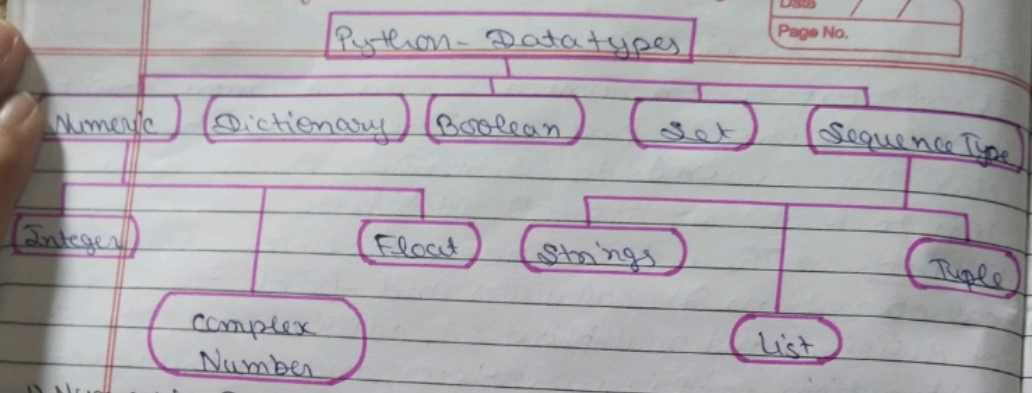
* Data Types

Python Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python Programming, Python data types are classes and variables are instances (objects) of these classes. The following are the standard or built-in data types in Python :

- Numeric
- Sequence Type
- Boolean
- Set
- Dictionary
- Binary Types (memoryview, bytearray)

*Float → precision upto (15 decimal places)
length of int - available memory, size complex no.

Date / /
Page No.



1) Numeric Data Types

The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number. These values are defined as Python int, Python float and Python complex classes in Python.

- Integers - This value is represented by int class. It contains +ve or -ve whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.
- Float - This value is represented by the float class. It is real number with a floating-point representation. It is specified by a decimal point. Optionally, the character e and E followed by a +ve or -ve integer may be appended to specify scientific notation.
- Complex Numbers - A complex number is represented by a complex class. It is specified as (real part) + (imaginary part) j. For example - 2+3j

Example → This code demonstrates how to determine the data type of variables in Python using the type() function. It prints the data types of three variables: a (integer), b (float) and c

Output :

2) Sequenc

The sequenc
collection
types. Seq
in an org
several
• Python
• Python
• Tuple

String Data
Strings in
Unicode cha
or more ch
quote or t
character da
of longer
Creating st
quotes, doub

multiline string { """ ... """ '''

Date / /
Page No.

String → consists of sequence of one or more characters (immutable)

(complex). The output shows the respective data type Python for each variable. Code ⇒

```
a = 5
print("Type of a:", type(a))

b = 5.0
print("\nType of b:", type(b))

c = 2+4j
print("\nType of c:", type(c))
```

Output: Type of a: <class 'int'>
Type of b: <class 'float'>
Type of c: <class 'complex'>

2) Sequence Data Types

The sequence Data Type in Python is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion. There are several sequence data types of Python.

- Python String
- Python list
- Tuple

String Data Type

Strings in Python are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in single quote, double-quote or triple quote. In Python, there is no character data type Python, a character is a string of length one. It is represented by str class.

Creating string: → Strings in Python can be created using single quotes, double quotes, or even triple quotes.

Date / /
Page No.

```
String1 = 'Welcome to the Geeks World'
print("String with the use of Single Quotes:")
print(String1)
String1 = "I'm a Geek"
print("\nString with the use of Double Quotes:")
print(String1)
print(type(String1))
String1 = '''I'm a Geek and I live in a world of "Geeks"'''
print("\nString with the use of Triple Quotes:")
print(String1)
print(type(String1))

String1 = '''Geeks
            For
            Life'''
print("\nCreating a multiline String:")
print(String1)
```

Output:

```
String with the use of Single Quotes:
Welcome to the Geeks World
String with the use of Double Quotes:
I'm a Geek
<class 'str'>
String with the use of Triple Quotes:
I'm a Geek and I live in a world of "Geeks"
<class 'str'>
Creating a multiline String:
Geeks
        For
        Life
```

List Data Type

Lists are just like arrays, declared in other languages

which is an ordered collection of data. It is very flexible as the items in a list do not need to be the same type

Creating List in Python

Lists in Python can be created by just placing the sequence inside the square brackets [].

Example → List = []
print ("Initial blank List:")
print (List)
List = ['GeeksForGeeks']
print ("\nList with the use of String:")
print (List)
List = ["Geeks", "For", "Geeks"]
print ("\nList containing multiple values:")
print (List[0])
print (List[2])
List = [['Geeks', 'For'], ['Geeks']]
print ("\nMulti-Dimensional List:")
print (List)

Output: → Initial blank List:
[]
List with the use of String:
['GeeksForGeeks']
List containing multiple values:
Geeks
Geeks
Multi-Dimensional List:
[['Geeks', 'For'], ['Geeks']]

Tuple Data Type

Just like a list, a tuple is also an ordered collection of Python objects. The only difference between a tuple

Date / /
Page No.

and a list is that tuples are immutable i.e tuples cannot be modified after it is created. It is represented by a tuple class.

Creating a Tuple in Python

In Python Data Types, tuples are created by placing a sequence of values separated by a 'comma' with or without the use of parentheses for grouping the data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, lists, etc.)

Note:→ Tuples can also be created with a single element but it is a bit tricky. Having one element in the parentheses is not sufficient, there must be a trailing 'comma' to make it a tuple.

Example →

```
Tuple1 = ()
print("Initial empty Tuple: ")
print(Tuple1)
Tuple1 = ('Geeks', 'For')
print("\nTuple with the use of String: ")
print(Tuple1)
list1 = [1, 2, 4, 5, 6]
print("\nTuple using List: ")
print(tuple(list1))
Tuple1 = tuple('Geeks')
print("\nTuple with the use of function: ")
print(Tuple1)
Tuple1 = (0, 1, 2, 3)
Tuple2 = ('python', 'geek')
Tuple3 = (Tuple1, Tuple2)
print("\nTuple with nested tuples: ")
print(Tuple3)
```

Output

3) Boo
Pytho
value
to Tu
are
can
and
by th
Examp

Output

4) Se
In Pyt
collectio
and e
elemen
may
Exampl

Output: Initial empty Tuple:
()
Tuple with the use of String:
('Geeks', 'For')
Tuple using List:
(1, 2, 4, 5, 6)
Tuple with the use of function:
('G', 'e', 'e', 'k', 's')
Tuple with nested tuples:
((0, 1, 2, 3), ('python', 'geek'))

3) Boolean Data Type
Python Data type with one of the two built in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). However non-Boolean objects can be evaluated in Boolean context as well and determined to be true or false. It is denoted by the class bool.

Example ⇒
```
print(type True))
print(type False))

print(type(true))
```

Output ⇒
```
<class 'bool'>
<class 'bool'>
```

4) Set Data Type
In Python Data Types, a Set is an unordered collection of data types that is iterable, mutable, and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

Example ⇒

```
set1 = set()
print("Initial blank set:")
print(set1)
set1 = set("GeeksForGeeks")
print("\nSet with the use of String:")
print(set1)
set1 = set(["Geeks", "For", "Geeks"])
print("\nSet with the use of List:")
print(set1)
set1 = set([1, 2, 'Geeks', 4, 'For', 6, 'Geeks'])
print("\nSet with the use of Mixed Values")
print(set1)
```

Output: Initial blank Set:
set()
Set with the use of String:
{'F', 'o', 'G', 's', 'r', 'k', 'e'}
Set with the use of List:
{'Geeks', 'For'}
Set with the use of Mixed Values
{1, 2, 4, 6, 'Geeks', 'For'}

5) Dictionary Data Type
A dictionary in Python is an unordered collection of data values, used to store data values like a map, unlike other Python Data Types that hold only a single value as an element, a Dictionary holds a key: value pair. key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a 'comma'.

Example →

Output: Em
{}
Dict
{1:
Dict
{1:
Dic
{1:
Dic
{1:
* Indent
Python in
before a
In anoth
same ind
code block

```
Dict = {}
print("Empty Dictionary:")
print(Dict)
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print("\nDictionary with the use of Integer keys:")
print(Dict)
Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}
print("\nDictionary with the use of Mixed keys:")
print(Dict)
Dict = dict({1: 'Geeks', 2: 'For', 3: 'Geeks'})
print("\nDictionary with the use of dict():")
print(Dict)
Dict = dict([(1, 'Geeks'), (2, 'For')])
print("\nDictionary with each item as a pair:")
print(Dict)
```

Output:

```
Empty Dictionary:
{}
Dictionary with the use of Integer keys:
{1: 'Geeks', 2: 'For', 3: 'Geeks'}
Dictionary with the use of Mixed keys:
{1: [1, 2, 3, 4], 'Name': 'Geeks'}
Dictionary with the use of dict():
{1: 'Geeks', 2: 'For', 3: 'Geeks'}
Dictionary with each item as a pair:
{1: 'Geeks', 2: 'For'}
```

* Indentation

Python indentation refers to adding white space before a statement to a particular block of code. In another word, all the statements with the same space to the right, belong to the same code block.

Date / /
Page No.

```
Statement
if condition:
        if condition:
                Statement
        else:
                Statement
Statement
```

How to interpreter visualises →

```
Code block 1 begins
Code block 1 continues
    Code block 2 begins
        Code block 3 begins
    Code block 2 continues
        Code block 3 continues
Code block 1 continues
```

Example of Python

- Statement (line 1), if condition (line 2), and statement (last line) belongs to the same block which means that after statement 1, if condition will be executed and suppose the if condition becomes False then the Python will jump to the last statement for execution.
- The nested if-else belongs to block 2 which means that if nested if becomes false, then Python will execute the statements inside the else condition.
- Statements inside nested if-else belong to block 3 and only one statement will be executed depending on the if-else condition.

Ex 1 ⇒

```
# Python program showing
# indentation

site = 'gfg'

if site == 'gfg':
    print('Logging on to geeksforgeeks...')
else:
    print('retype the URL.')
print('All set!')
```

Output: Logging on to geeksforgeeks...
All set!

Ex 2 ⇒

Output ⇒

* Taking
Developers
either to g
Most prog
of asking
While Pyt
to read
• input(-f
• raw_in
input(): T
the user a
the return
does not
the comp
provides a
taken fr
function
waits fo
enter, the
the user
Syntax:

Date / /
Page No.

Ex 2 ⇒ i = 1

```
while (i<=5):
    print(i)
    i = i + 1
```

Output ⇒ 1
2
3
4
5

* Taking input in Python

Developers often have a need to interact with users, either to get data or to provide some sort of result. Most programs today use a dialog box as a way of asking the user to provide some type of input. While Python provides us with two inbuilt functions to read the input from the keyboard.

- input(prompt)
- raw_input(prompt)

input(): This function first takes the input from the user and converts it into a string. The type of the returned object always will be <class 'str'>. It does not evaluate the expression it just returns the complete statement as string. For example, Python provides a built-in function called input which taken the input from the user. When the input function is called it stops the program and waits for the user's input. When the user presses enter, the program resumes and returns what the user typed.

Syntax:

* Existing Python Structure ->
Modules →
Statements →
Expression →
Object →

Date / /
Page No.

inp = input ('STATEMENT')

Example:

1. >>> name = input('What is your name?\n') # \n ---> newline ---> It causes a line break
>>> What is your name?
Ram
>>> print(name)
Ram

---> comment in python

ode ⇒ # Python program showing
a use of input()

val = input("Enter your value: ")
print(val)

Output: Enter your value: 123
123
>>>

Taking string as an input:

name = input('What is your name?\n') # \n ---> newline ---> It causes a line break
print(name)

Output: What is your name?
Ram
Ram

plete Code ⇒ # Program to check input
type in Python

num = input("Enter number: ")
print(num)
name1 = input("Enter name: ")
print(name1)

Date / /
Page No.

```
# Printing type of input value
print (" type of number ", type(num))
print (" type of name ", type(name1))
```

Output :→ Enter number : 123
123
Enter name : geeksforgeeks
geeksforgeeks
type of number <class 'str'>
type of name <class 'str'>
>>>

raw_input() : This function works in older version (like Python 2.x). This func^n takes exactly what is typed from the keyboard, converts it to string and then returns it to the variable in which we want to store it.

Example :→

```
# Python program showing
# a use of raw_input()

g = raw_input("Enter your name :")
print g
```

Output : Enter your name : geeksforgeeks
geeksforgeeks
>>>

* Print Output

The 'print()' function is used to output data to the console.

Example :

```
print ("Hello, World!")
```

* Type Conversions

The act of changing an object's data type is known

Date / /
Page No.

as type conversion. The Python interpreter automatically performs implicit Type Conversion. Python prevents Implicit Type conversion from losing data.
The user converts the data types of objects using specified functions in explicit type conversion, sometimes referred to as type casting. When type casting, data loss could happen if the object is forced to conform to particular data type

Type conversion

Implicit Type conversion | Explicit Type conversion

Implicit Type Conversion → In implicit type conversion of data types in Python, the Python interpreter automatically converts one data type to another without any user involvement. To get a more clear view of the topic see the below examples →

```
x = 10
print("x is of type:", type(x))
y = 10.6
print("y is of type:", type(y))
z = x+y
print(z)
print("z is of type:", type(z))
```

Output :→ x is of type: <class 'int'>
y is of type: <class 'float'>
20.6
z is of type: <class 'float'>

Explicit Type Conversion → In Explicit Type conversion in Python, the data type is manually changed by the user as per their requirement. With explicit type conversion, there is a risk of data loss since we are forcing an

Date / /
Page No.

expression to be changed in some specific data type. Various forms of explicit type conversion are: int(a, base), float(). Example →

```
# initializing string
s = "10010"
# printing string converting to int base 2
c = int(s,2)
print("After converting to integer base 2 : ", end = " ")
print(c)
# printing string converting to float
e = float(s)
print("After converting to float : ", end = " ")
print(e)
```

Output: → After converting to integer base 2 : 18
After converting to float : 10010.0

* Dynamic and Strongly Typed language

• Dynamic Typing: In Python, the type of a variable is determined at runtime and can chang dynamically.

```
Eg → x = 10    # x is an integer
     x = "Hello" # Now x is a string
```

• Strongly Typed: Python enforces type constraints. Operations on incompatible type result in errors Eg →

```
x = 10
y = "20"
# print(x+y) # This will raise a TypeError
```