WEEK 1 ~PYTHON BASICS~

What is Python ?
→ Python is a programming language.

Who created Python ?
→ Python was created by Guido van Rossum and it was first implemented in 1989.

What is Python used for ?
→ 1. Web development
2. Machine Learning
3. Data science
4. Scripting
5. And many more

Latest Version of Python → python 3

Python Interpreter (Compiler)
→ A python interpreter is a program that directly executes python codes.

Python console (Shell)
→ Python console also known as shell allows you to execute python codes line by line.
(Simply, its an output screen)

Python Syntax

Syntax is a set of rules followed when writing python programs

Python Indentation

→ Indentation in python is very important.

→ Indentation indicates a block (group) of statements.

→ Tabs or leading whitespaces are used to compute the indentation levels of a line.

→ It depends on you whether you want to use tabs or whitespaces

For example,

```
if x > 4:
    print("Hello")
```

NOTES : The no. of whitespaces also depends on you, but it has to be at least one.

Indentation Error

→ In python, a block of statements needs to have the same indentation level.

→ This example will produce an error, because the

third line does not have the same indentation as the second line.

```
if 5>4:
x = "Hello World"
  print(x)
```

Ending A Block of Statements

A block (group) of statements ends with the next unindented line.

```
if 5>4:
    x = "Hello!"
    print(x)
print("The block of statement ends here!")
```

Python Comments

→ Comments in python are commonly used to clarify or explain codes.

→ Comments are not interpretted by python. Meaning, comments will not be executed.

CODE →

```
#this is a comment
x=3
y=4
print(x+y)  #add x and y
```

By Gagneet Kaur

- Single Line Comment

A single line comment starts with the hash # character.

CODE ⟹

```
# this is a single line comment
print ("Hello")
```

- Multiple Line Comments

Just use multiple single line comments.

```
# first comment
# second comment
print ("Hello")
```

By Gagneet Kaur

(Displaying Output in Python) PRINT ( )

- To display an output in python, you need to use a special keyword print

'Use print( ) function to display output'

- We can print a string, number or character using this function.

Code → print ('I am a girl')
Output → I am a girl

- The print ( ) function can also be used to print two objects using 'comma' (,).

Code → print ('The sum is', 3+1)
Output → The sum is 4

NOTE : for string datatype, we need to use quotes around the text in print ( ) fn.
If you start with single quotes then end with single quotes only and same for double quotes. Mixture of single & double quotes will show an error. for example,

```
print ('Hello')
print (" Hello ")
print ('Hello")
```

Here first and second statements are correct. But third one is wrong.

Precedence & Associativity
OF OPERATORS IN PYTHON

PRECEDENCE

OPERATORS	Meaning
( )	Parentheses
**	Exponent
* , / , // , %	Multiplication , Division , Floor Division , Modulus
+ , -	Addition , Subtraction
== , != , > , >= , <=	Comparisons
not	Logical NOT
and	Logical AND
or	Logical OR

The operator precedence in Python is listed in the above table. It is in descending order (upper group has higher precedence than the lower ones).

ASSOCIATIVITY

→ We can see in the above table that more than one operator exists in the same group. These operators have the same precedence.

→ When two operators have the same precedence, associativity helps to determine the order of operations.

→ Associativity is the order in which an expression is evaluated that has multiple operators of the same precedence. Almost all the operators have left to right associativity.

→ For Example,

* and // have the same precedence. Hence, if both of them are present in an expression, the left one is evaluated first.

→ For the exponential operator (**), associativity is from right to left.

Example,

```
2** 3** 8 **4
```

→ python will parenthesize the above codes as:

```
(2** (3** (8**4)))
```

→ Also, the not operator has right to left associativity.

VARIABLES

→ Variable = vary + able
variable is something that could vary.

> • Use a variable to write something that could change.

• As a programmer, you will write variables everyday and its very simple to write a variable. You will need just 3 things :

1. A name (variable)
2. An equal sign (=)
3. A value (literal)

Variable → which varies
literals → they are constant values and are only used at right hand side of equal sign.

EXAMPLE : age = 20

name (variable) — equal sign — value (literal)

→ You can set different values multiple times to a variable. Only the last value will stay.

```
Code →  age = 12
        age = 18
        age = 23
        print(age)
```

Output → 23

→ A variable is like a bucket, where every bucket has a name or lable and a value inside it.

* Rules for naming variables

- A variable name should start with a letter or underscore (_)
- A variable name can't start with a number.
- A variable name is case sensitive.
- A variable name should only contain underscores (_) and alphanumeric characters.

→ When naming a variable with a long name, use underscores to separate words.

Example, my_favourite_fruit = "mango"

→ You can also use the camelCase format where every first letter of words are capitalized.

myFavFruit = 'mango'

USER INPUT

→ To ask any information in python prog. lang., just type input().

→ While asking for user input, you can add extra info to tell the user what information they should provide.

```
input ('What is your name?')
```

→ Input in variable :
You can also store user input in a variable and then use that variable.

```
Code → name = input ("What is your name ?")
       print ("Name :" + name)

Output → What is your name?
         Gagneet Kaur
         Name : Gagneet Kaur
```

→ Data type of input() :
By default, user input is taken as a string. When the user inserts string or number, both are taken as string.
We can convert into a number by using special keyword int (for number) or float (for decimal).

```
x = int ( input ('What is your age'))
g = float ( input ('Value of pi :'))
```

To know the datatype of a variable, write the full code — `print(type(variable))` {USE the keyword type}

classmate
Date May 3, 2021
Page 0

DATA TYPES

Type Conversion
Type Casting

(1) String

→ When the value of a variable contains text (one or more words), it is called string.

→ While writing a string type variable, the value of the variable has to be inside double quotes or single quotes.

for example, name = 'Gagneet Kaur'
sports = "Basketball and Athletics"

* To combine strings, use plus sign (+) →

CODE,

```
a = 'Hello'
b = 'World'
print(a+b)
```

OUTPUT

HelloWorld

(2) Numbers

In python, there are two basic types of numbers and they are called integer and floating point numbers.

- An Integer does not have decimals.

```
#x is an integer
x = 21
print(x)
```

- A floating point number has decimals.

```
#x is a floating point number
x = 21.33
print(x)
```

★ INT vs FLOAT

for age, you will use int datatype
for price or rate, you will use float.

But in some cases, you might not be sure whether user will insert an integer or a float no. In those cases, use float to be on safe side.

Besides, float is more powerful. It can handle both. However, int can't handle a no. with a fraction or decimal.

(3) Booleans

→ A variable whose value could be either True or False is called boolean type variable.

→ Writing a boolean type variable is very simple.

```
if yesOrNo = True
    print(yesOrNo)
```

(equal sign; Boolean value (True or False); a variable)

→ NOTE : True or False starts with capital letter.

for example, isRich = True
isScared = False

→ When comparing two values, python returns a Boolean.

→ We don't put double quotes while writing Boolean values (True or False), because they are specially recognised keywords used for Boolean type.

(4) Lists

→ A list is ~~is~~ an ordered collection of data.

→ It can contain strings, numbers, etc.

→ Lists are written with square brackets. ([]).

→ The values (also called elements) in a list are separated with commas (,).

```
myList = [2, 4, 6, 8, 'yes', 'no']
print(myList)
```

Getting the Type Of A Variable

We get the specific data type of a variable using the type() function.

CODE :

```
a = 3
b = "Hi"
c = True
d = [1, 3, 5]
print(type(a), type(b))
print(type(c), type(d))
```

Output

```
<class 'int'> <class 'str'>
<class 'bool'> <class 'list'>
```

Type Conversion or Type Casting

In python, if you try to concatenate or combine a string and a number, you will get an error.

To prevent the error, we should first convert the type of the number to string (str).

- **Convert to String**

The str() function returns the string version of the given object.

CODE :

```
x = 21
x = str(x)
text = 'My fav no. is' + x
print(text)
```

- **Convert to integer**

Use int() fn

CODE :

```
x = 4.55
x = int(x)
print(x)
```

Output 4

- **Convert to Float**

Use float() fn.

CODE:
```
x = 4
x = float(x)
print(x)
```
Output → 4.0

- Convert to Boolean → use bool() fn

CODE:
```
x = 10
yx = 'Hello'
z = ''
x = 0
print(bool(x))
print(bool(yx))
print(bool(z))
print(bool(x))
```
Output:
True
True
False
False

OPERATORS

- What are operators?

→ Operators are symbols that perform operations on operands.

→ Operands can be variables, strings, numbers, booleans etc.

(1) Arithmetic Operators

Arithmetic operators perform mathematical operations on its numerical operands.

In python, there are 7 basic arithmetic operators:

OPERATOR	DESCRIPTION
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
%	Remainder
//	Floor Division

(a) → Addition Operator : · It returns the sum of its numerical operands.

CODE → x = 10
y = 5
print (x + y)

Output → 15

· It can also be used to concatenate or join strings.

CODE → x = "Hello" + "World"
print (x)

Output → HelloWorld

(b) Subtraction Operator : It returns difference of its numerical operands.

CODE → x = 10
y = 5
print (x - y)

Output → 5

(c) Multiplication Operator

It returns the product of its numerical operands.

CODE :
```
x = 10
y = 5
print (x*y)
```
OUTPUT : 50

(d) Division Operator

It returns the quotient of its numerical operands.

NOTE: The quotient is always a floating point no.

CODE,
```
x = 10
y = 5
print (x/y)
```
OUTPUT, 2.0

(e) Exponentiation Operator

It raises the left operand to the power of the right operand.

CODE :
```
#same as 2* 2* 2
x = 2 ** 3
print (x)
```
Output : 8

(f) Remainder Operator

The remainder operator, also known as the modulus operator returns the remainder after dividing the left operand by the right operand.

In the following example, 5 is divided by 2, therefore the quotient is 2 and the remainder is 1.

CODE: `x = 5 % 2`
`print (x)`

Output: 1

(g) Floor Division Operator

The floor division operator (//) rounds down the quotient of its numerical operands to the nearest whole no.

In the foll. example, the quotient is 2.5. The result of rounding down 2.5 is 2.

CODE: `x = 5 // 2`
`print (x)`

OUTPUT: 2

✱ OPERATOR SEQUENCE

- Operator sequence describes the order of performed operations in an arithmetic expression.

code →
```
x = 10 + 3 * 5
print (x)
```
Output → 25

- The result is 25 because multiplication is performed first before addition.
  $3 * 5 = 15$, and then $10 + 15 = 25$

- Python follows the DMAS pattern.
  Division >> Multiply >> Add >> Subtract

- But what if we need a certain operation to be performed first ?

We can use grouping to do that, grouped expressions are performed first before the others.

Use parentheses ( ) to group expressions.

CODE :
```
x = (10+3) * 5
#returns 65 because 10+3 = 13, the 13*5 is 65
print (x)
```

OUTPUT : 65

---

(2) Comparison Operators

→ A comparison operator compares its operands and returns a Boolean value based on whether the comparison is True or False

→ It can be used to compare strings, numbers, Booleans & other objects.

OPERATOR	DESCRIPTION
==	Equality
!=	Inequality
>	Greater than

<	less than
>=	Greater than or equal to
<=	Less than or equal to

(a) Equality : The equality operator (==) compares two values and returns True if the operands are equal, otherwise returns False.

CODE : print(5 == 5) OUTPUT : True

CODE : print("python" == "python") OUTPUT : True

(b) Inequality : The inequality operator (!=) compares two values and returns True if the operands are NOT equal, otherwise returns False.

```
CODE : print(10 != 5)        OUTPUT : True
       print('s' != 's')              False
       print(2 != 2)                  False
```

(c) Greater than : The greater than operator (>) returns True if the left operand is greater than the right operand, otherwise returns False.

```
CODE : print(8 > 4)          OUTPUT : True
       print(5 > 10)                  False
```

(d) Less Than : The less than operator (<) returns True if the left operand is less than the right operand, otherwise returns False.

CODE: print (10<15) Output: True
print (20<10) False

(e) Greater than or equal to (>=) It returns True if the left operand is greater than or equal to the right operand, otherwise returns False.

CODE: print (8>=4) Output: True
print (8>=8) True
print (5>=10) False

(f) Less than or equal to (<=) : It returns True if the left operand is less than or equal to the right operand, otherwise returns False

CODE: print (10<=15) Output: True
print (10<=10) True
print (20<=10) False

---

(3) Logical Operators

→ Logical operators are commonly used with Booleans.
→ In Python, there are 3 logical operators.

OPERATOR	DESCRIPTION
and	Logical and
or	Logical or
not	Logical not

(a) Logical And : The logical and operator returns True if both operands are True.

CODE :
```
print (True and True)
print (True and False)
print (False and False)
```
Output :
```
True
False
False
```

→ Logical operators are commonly used on expressions.

CODE:
```
x = (5 == 5) and (4 == 4)
print (x)
```
Output : True

→ In the foll. example, the expression returns False because one of the operands is False.

CODE :
```
x = 4
expr = (x > 3) and (8 < x)
print (expr)
```
Output: False

(b) Logical OR : The logical OR operator returns True if either of the operands is True.

CODE:
```
print (True or True)
print (True or False)
print (False or False)
```
Output :
```
True
True
False
```

(c) Logical NOT : The logical not operator returns True if the operand is False.

CODE:
```
print (not (False))
print (not (True))
```
Output :
```
True
False
```

CODE:
```
expr = not (10>20)
print (expr)
```
Output: True

CODE
```
expr = not (10==10)
print (expr)
```
Output: False

STRINGS

Strings in python, are sequences of characters or simply text.

Concatenating Strings

Concatenating strings simply means combining or adding strings together.

To combine strings, use the plus sign (+).

CODE:
```
a = 'Hello'
b = 'World'
text = a+b
print (text)
```
Output: HelloWorld

Accessing Parts of a String

In python, we can access a single character or a range of characters from a string.

(1) Indexing: To access a single character, use indexing.

→ Indexing uses square brackets ([]) to access characters.
→ 0 represents the first character, 1 represents second character and so on.

CODE:
```
text = "Hi everyone"
print (text[0])
print (text[1])
```
Output: H
i

→ While -1 represents the last character, and -2 represents the second last character.

CODE:
```
text = "Hi everyone"
print (text[-1])
print (text[-2])
```
Output: e
n

(2) Slicing: To access a range of characters, use slicing. Slicing also uses square brackets.

→ The square brackets can contain 2 integers separated by a colon.
→ The first integer is the start index, the second integer is the end index (exclusive)

CODE:
```
text = "Python Lover"
print (text[2:5])
```
Output: tho

→ Even though index 5 represents the letter n in ~~our~~ example above, it was still not printed because when slicing, end index is not included.

→ To slice from a specific position ~~to~~ until the end

of the string, don't specify the second integer.

CODE:
```
text = "Python Lover"
print(text[2:])
```
Output: thon Lover

Getting the Length of A String

→ The length of the string is the number of characters it contains.

→ The len() function returns the length of a string.
→ It takes one parameter, the string.

CODE:
```
text = 'Hello'
print(len(text))
```
Output: 5

Replication

It replicates the string.

CODE:
```
text = 'Gagan'
x = text * 4
print(x)
```
Output: GaganGaganGaganGagan

CODE:
```
text = 'Gagan'
x = text[0] * 4
print(x)
```
Output: GGGG

String Comparison

CODE:
```
x = 'India'
print ( x == 'India')
print ( x == 'india')
```

Output: True
False

CODE:
```
print ('apple' > 'one')
# it checks if first letter ~~come~~ of each operand (string) and
  compares if a is greater than o or not, which is
  False, thus it will return False
print ('four' < 'ten')
```

Output: False
True

CODE:
```
print ('ab' < 'az')
print ('abcdef' < 'abcde')
```

Output: True
False

SUMMARY OF WEEK 1

- print ( ) → output / display
- input ( ) → to get input from user

- datatypes :
  1. string str( )
  2. integer int( )
  3. floating point no. float( )
  4. Booleans True , False
  5. Lists l = [ ]

- type ( ) → to get the datatype of variable

- operators →
  1. Arithmetic (+ , - , * , ** , / , // , %)
  2. Comparison (> , < , >= , <= , == , !=)
  3. Logical (and , or , not)

- strings :
  1. concatenating (use + sign)
  2. indexing (text[0] , text[1] ...)
  3. slicing (text[2:5] , text[3:] ...)
  4. len ( ) → get length of string
  5. replication (*)
  6. string comparison