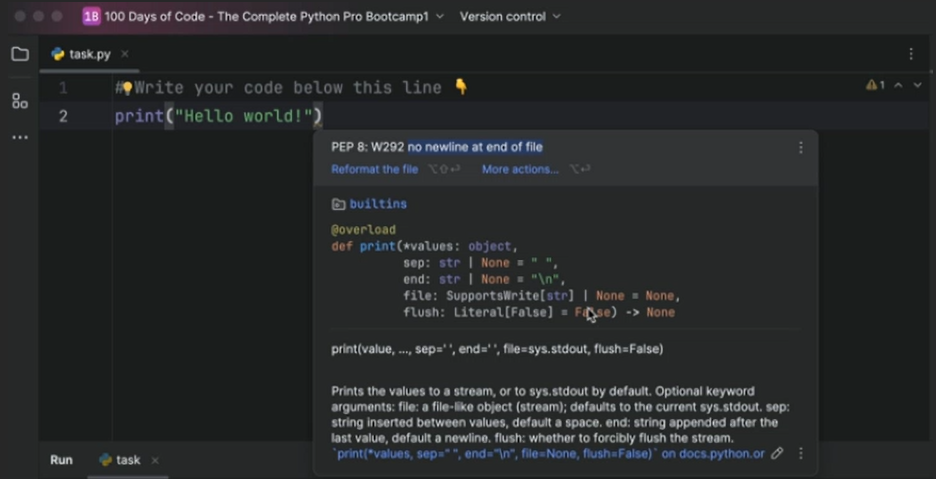
**Python YAY**

**Print Function:**

print(“Hello World!”) #code

>>> Hello World! # output

The piece of text between the inverted commas are called string.



It is a good practice to leave an empty newline at the end of the python file.

1. “\n” can be used to end a line in string

Example:

print(“Hello! \nWorld!”)

>>> Hello!

World!

1. Two strings can be combined using + operator (This is called sting concatenation)

Example:

print(“Hello ” + ”World!”)

>>>Hello World!

**Input Function:**

An input function can be used to take input from the user into a variable to directly into a function etc

Example:

input(“What is Your Name ? : \n”) #code

>>>What is Your Name ? : #output

Yash Sharma 🡪 Input by the user

^This data can be stored into a variable as:

Name = input(“What is your name ? : ”) #Code

***Program:***

Name = input(“What is your name?: \n”)

print(“Hello! ” + Name)

***OUTPUT:***

>>>What is your name?:

YASH SHARMA

Hello! YASH SHARMA

This whole program can be written in a single line too:

print(“Hello! ” + input(“What is your name?: \n”))

This would have the same output as the above.

**Python variables:**

Variables can be considered as named storage location in a computer’s memory that can store data. This stored data can be accessed anywhere throughout the code. As its name suggests its value can be varied i.e. it can be changed.

It is a good practice to name variables as words instead of characters as it increases readability of code.

**Variable Naming:**

|  |  |
| --- | --- |
| **Incorrect** | **Correct** |
| User name, num 1, length of side | Can’t have gap between words, underscore can be used in place of space |
| 1name | First character can’t be an integer. |
| input , print , len | Cant have keywords as variables. |
| a@ , username@ | Variables cant have special characters |

**len():**

It can be used to print the length of the string.

Example:

A = “hello”

Length = len(A)

print(Length)

>>>6

**Data Types:**

We can use type() to find the data type.

Ex: print(type(“hello”))

>>><class ‘str’>

Below are the data types present in Python.

1. String: It can be referred as an array of characters.

Ex: “hello world”

Since it is a string of characters, we can pull out its characters individually.

Example:

print(“Hello”[0])

>>>H

1. Integers: It contains integers ranging from negative infinity to positive infinity.

Ex: 1,2,3,4

To increase the code readability, we put underscore between numbers while printing as it is.

Ex: print(100\_000\_000)

>>>100000000

1. Floats: It contains floating numbers i.e. number with a point in it.

Ex: 1.0, 2.3, 1.5

Same can be done as above:

Ex: print(100\_000.56)

>>>100000.56

1. Boolean: It contains True and False only.

TypeError object represents an error when an operation could not be performed, typically (but not exclusively) when a value is not of the expected type

Ex: len(12345)

Here an integer value is passed where a string is expected.

**Type Conversion:**

Type conversion can be carried out using certain functions.

1. int(): It changes data type to an integer value.

Ex: int(“123”) 🡪 123

Int(1.2) 🡪 1

Int(True) 🡪1

1. float(): It changes data type to a floating value.

Ex: float(“1.2”) 🡪 1.2

float(“1”) 🡪 1.0

float(12) 🡪 12.0

1. str(): it takes the character present in the data type and store them in a array of character.

Ex: str(True) 🡪’True’

str(1) 🡪 ‘1’

str(1.2) 🡪 ‘1.2’

1. bool() : It gives output as False only if the passed value is 0. It gives True otherwise.

Ex: bool(123) 🡪 True

bool(-123) 🡪 True

bool(0) 🡪 False

bool(“w”) 🡪 True

bool(“”) 🡪 False

Note: For type conversion of string to integer or float the string should only contain numbers i.e. characters ranging from 0 to 9 (0-9).

**Arithmetic Operators in Python:**

1. (+) : Does the operation of Addition.

Ex: 1+2 = 3

1. (-) : Does the operation of Subtraction.

Ex: 2-3 = -1

1. (\*) : Does the operation of Multiplication.

Ex: 3\*2 = 6

1. (/) : Does the operation of Division.

Ex: 3/2 = 1.5

2/2 = 2.0

1. (//) : Does the operation of Floor Division.(Floors the division)

Ex: 3//2 = 1 (3/2 = 1.5 , after flooring we get 1)

1. (%) : Does the operation of Modulus. (Remainder of the division)

Ex: 7%5 = 2

1. (\*\*) : Does the Exponentiation of a number.(To the power)

Ex: 3\*\*2 = 9

**NOTE : Python follows PEMDAS order during evaluation.**

**Parenthesis 🡪 Exponents 🡪 Multiplication 🡪 Division 🡪 Addition 🡪 Subtraction**

We can use round() to round of a floating value to certain decimal.

Ex: round(3.2) 🡪 3

round(3.6) 🡪 4

round(3.879976 , 2) 🡪 3.88

**Assignment Operators In Python:**

|  |  |
| --- | --- |
| Operator | Same As |
| = | X = 5 |
| += | X = X + 5 |
| -= | X = X – 5 |
| \*= | X = X \* 5 |
| /= | X = X/5 |
| %= | X = X%5 |
| //= | X = X//5 |
| \*\*= | X = X\*\*5 |

**F STRINGS:**

We can use f strings to add values directly into the strings without having to separate different parts of string. (Just like in C we use %s, %f, %d)

***Program:***

score = 45

print(f“You scored: {score}”)

>>>You scored: 45

**IF ELSE STATEMENT:**

It is a conditional statement that executes different blocks of code based on whether a condition is true or false.

An if else statement works as follows:

If (condition):

Code

elif (condition):

Code

else:

Code

Another if else statement inside an if else statement is called a nested if else statement.

**Comparison Operators:**

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| > | Greater than |
| < | Lesser than |
| >= | Greater than or equal to |
| <= | Lesser than or equal to |
| != | Not equal to |
| == | Equal to |

**Logical Operators:**

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| and | Produces 1 when all the inputs are 1 |
| or | Produces 1 when any one input is 1 |
| not | Compliment of the input |

**MODULE:**

A module is a piece of built in code that can be called using different keywords associated with the imported module.

These modules help in reducing the size of the code and increase readability of code.

**Random Module:**

We import the function using import keyword as following:

import random

we can also give name to this module for calling by:

import random as r

**Random Integer Generation:**

1. random.randrange(x) or r.randint(x,y): a random number (N) is generated such that:

in first case: 0 <= N < x

in second case: x <= N < y

1. random.randint(x) or r.randint(x,y): works the same as randrange but includes both the parameters in the range i.e. the generated number N will be such that

x <= N <= y

**LIST:**

Lists are one of the ways to store multiple data in single variable. In technical terms list is a mutable data structure.

It is denoted by:

X = [1,2,3,4] X = [‘pear’ , ’apple’] X = [1,2,’hello’]

**FUNCTIONS RELATED TO LISTS:**

1. list.append(x) : appends (adds) an element x into the list.
2. list.pop(x) : removes the element present at the x index.
3. list.clear() : removes all the elements in the list.
4. list.count(x) : returns the number of x elements present in the list.
5. list.insert(x,y) : adds the elements y at x index.
6. list.sort(reverse = False) : sorts the list in ascending order.
7. list.reverse() : reverses the order of the list.
8. list.remove(x): removes the first x element in the list.
9. list.index(x) : returns the first index of x present in the list.
10. list.extend(list2) : add the list2 at the end of list.

A list inside of another list is called a nested list.

This can be denoted by: [1,2,3,[4,5,6]]

**While Loop:**

A while loop runs until the condition specified is true.

For example:

while True:

print(“Working!”)

>>>Working

Working

Working

Working

Working

.

.

.

.

**USER-DEFINED FUNCTIONS:**

A user defined function works the same as the built -in functions but instead this function is defined by the user before calling.

A user defined function can be called using def keyword.

**Example:**

def main():

\*\*\*\*Code

This function can be called by its name.

main()

the code has to be indented so that it stays inside the main function.

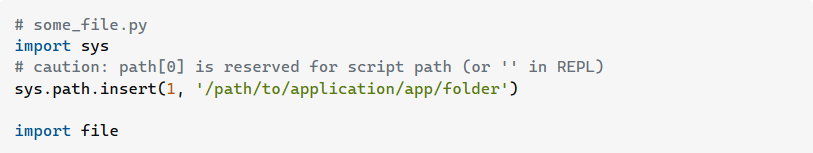
This indentation consists of 4 spaces. But the code works even if the spaces are less than 4, there should be at least 1 space.

We can create user defined modules and import them into our project using import keyword. In order to create this user defined module, we have to define a function in a separate python file. We can now import this module using import keyword and the name of the file. We can call the function using its name.

Example:

|  |  |
| --- | --- |
| Current File | Module File |
| Import land as l  print(l.bing())  >>>Ling | def land(){  return “Ling”  } |

We can import file from different folder by importing sys.



We can add arguments into the function by specifying a variable in the bracket after the name while function is defined.

Example:

def a(x):

print(x)

print(a(1))

>>>1

|  |  |
| --- | --- |
| Position Arguments | Keyword Arguments |
| def a(x,y,z):  print(x)  print(y)  print(z)  a(1,2,3)  >>>1  2  3 | def a(x,y,z):  print(x)  print(y)  print(z)  a(y = 2,x=1,z =3)  >>>1  2  3 |

**ord() and chr():**

This function can be used to convert characters into its ASCII value.

print(ord(‘a’)

>>>97

Numbers can be converted to characters by using chr() function.

print(chr(98))

>>>b

**DICTIONARY:**

A dictionary in Python is a built-in data type that stores data in key-value pairs. It is an unordered, mutable, and indexed collection. Dictionaries are written with curly brackets, and have keys and values.

Ex: x = {“key1”:”value1”,”key2”:”value2”}

We can call values using its key

Ex: print(x[“key1”])

>>>value1

We can change the values of the key using below method

x[“key1”] = “value69”

Note: when we run a for loop in dictionary the for loop loops through its keys not its value

Example:

x = {“key1”:”value1”,”key2”:”value2”}

for i in x:

print(i)

>>>key1

>>>key2

**title():**

This function changes the first letter to uppercase and others to lowercase in a string.

**Return In Function:**

A**return statement** is used to end the execution of the function call and it “returns” the value of the expression following the return keyword to the caller. The statements after the return statements are not executed. If the return statement is without any expression, then the special value None is returned. A **return** **statement** is overall used to invoke a function so that the passed statements can be executed.

Ex.

def main(x):

return x\*5

print(main(10))

>>>50

**Commenting:**

This is a piece of text that is not executed when the code runs.