In python the space between the variables is automatically added which can be explained by the example given below:

Ex:

name\_person = “Steven”

print(“Hey my name is”, name\_person)

# Output of the program is Hey my name is Steven

Here the space before the name Steven is automatically added.

You cannot use reserved words as variables names/identifiers. The extension of the python files is “.py”.

Interactive v/s Script Python: A) In Interactive python you type directly to python one line at a time and it responds. B) In Script you enter a sequence of statements in to a file using a text editor and tell python to execute the statements in the file.

The way to print the variable inside a string is to put them inside the {} braces. It can be explained by the example given below:

Ex:

my\_name = ‘Zed A. Shaw’

my\_age = 35

my\_height = 74

my\_weight = 180

my\_eyes = ‘Blue’

my\_teeth = ‘White’

my\_hair = ‘Brown’

print(f”Let’s talk about {my\_name}.”)

print(f”He’s {my\_height} inches tall.”}

print(f”He’s {my\_weight} pounds heavy.”}

print(“Actually that’s not too heavy.”}

print(f”He’s got {my\_eyes} eyes and {my\_hair} hair.”)

print(f”His teeth are usually {my\_teeth} depending on the coffee.”)

total = my\_age + my\_height + my\_weight

print(f”If I add {my\_age}, {my\_height}, and {my\_weight} I get {total}.”)

Output:

Let’s talk about Zed A. Shaw.

He’s 74 inches tall.

He’s 180 pounds heavy.

Actually that’s not too heavy.

He’s got Blue eyes and Brown hair.

His teeth are usually White depending on the coffee.

If I add 35, 74, and 180 I get 289.

The variable name in python should always start with a character. So 1 is not a valid variable name while a1 is a valid variable name. We can use the round() function to round the floating point number.

In python it is typical to use single-quotes for any short strings like ‘a’ or ‘snow’.

Escape Sequences:

|  |  |
| --- | --- |
| Escape | What it does. |
| \\ | Backslash (\) |
| \’ | Single-quote (‘) |
| \” | Double-quote (“) |
| \a | ASCII bell (BEL) |
| \b | ASCII backspace (BS) |
| \f | ASCII formfeed (FF) |
| \n | ASCII linefeed (LF) |
| \N {name} | Character named name in the Unicode database (Unicode only) |
| \r | Carriage return (CR) |
| \t | Horizontal tab (TAB) |
| \uxxxx | Character with 16-bit hex value xxxx |
| \Uxxxxxxxx | Character with 32-bit hex value xxxxxxxx |
| \v | ASCII vertical tab (VT) |
| \ooo | Character with octal value 000 |
| \xhh | Character with hex value hh |

**Typecasting:**

1. Typecasting the input to Integer: There might be condition when you might require integer input form user/console, the following code takes two input(integer/float) from console and typecasts them to integer then prints the sum.

Ex:

Num1 = int(input())

Num2 = int(input())

#printing the sum in integer

print(Num1 + Num2)

1. Typecasting the input to Float: To covert the input to float the following code will work out.

Ex:

Num1 = float(input())

Num2 = float(input())

#printing the sum in float

print(Num1 + Num2)

1. Typecasting the input to String: All kind of input can be converted to string type whether they are float or integer. We make use of keyword str for typecasting.

Ex:

String = str(input())

#output

print(String)

Reading and Writing Files:

1. Read: Reads the contents of the file. You can assign the result to a variable.
2. Readline: Reads just one line of a text file.
3. Truncate: Empties the file. Watch out if you care about the file.
4. Write(‘stuff’): Writes “stuff” to the file.
5. Seek(0): Moves the read/write location to the beginning of the file.

When we want to interrupt or terminate the execution of the program while it is running in the python terminal then we should use combination of **CTRL + C** to execute it.

If we want the input() function to do nothing then we should press the **ENTER** button to execute the next line in our program.

There are many different ways to check if a file exists or not:

1. The exists() method in the os.path. Ex: os.path.exists(‘file\_ex.txt’)

If you want to ensure that a given path points to a file and not to a directory, you can use the os.path.isfile() function. Ex: os.path.isfile(‘file\_ex.txt’)

1. If the file exists the open call will complete successfully and return a valid file handle. If the file does not exist however, a FileNotFoundError exception will be raised.

Ex: try

F = open(‘myfile.txt’)

F.close()

except FileNotFoundError:

print(“File does not exist”)

1. Now the same “just attempt to open it” technique also works for ensuring a file is both readable and accessible. Instead of watching for FileNotFoundError exceptions you will want to look out for any kind of IOError:

Ex: try

F = open(“myfile.txt”)

F.close()

except IOError:

print(“File is not accessible”)

print(“File is accessible”)

1. If you frequently use this pattern you can factor it out into a helper function that will allow you to test whether a file exists and is accessible at the same time:

Ex: def is\_accessible(path, mode=’r’):

try:

f = open(path, mode)

f.close()

except IOError:

return False

return True

Alternatively, you can use the os.access() function in the standard library to check whether a file exists and is accessible at the same time. This would be more similar to using the os.path.exists() function for checking if a file exists.

Using open() and a try…except clause has some advantages when it comes to file handling in Python. It can help you avoid bugs caused by file existence race conditions.

It is recommended to use the file handling exception way to treat whether a file exists or not because, it will check right away if the file exists or not. While in using the exists() function if we check whether a file exists or not then we check first that the file exists() and it return True, but after executing the next line if the file is deleted then we will get an IOError which says that it is better to just directly open the file and using exception to handle whether the file exists or not.

In python when you have bytes and need a string decode bytes. When you have a string and need bytes encode strings.

Syntax: range([start], stop [, step])

Parameters:

Start: (optional) Starting point of the sequence. It defaults to 0.

Stop: (required) Endpoint of the sequence. This item will not be included in the sequence.

Step: (optional) Step size of the sequence. It defaults to 1.

The type() function is used to find the type of the input given to the type function.

Ex: type(eee)

<class’str’>

type(1)

<class’int’>

To convert an integer to floating point you can do ex: float(99) and it will be converted to 99.0

Integer division produces a floating point result.

String conversions: You can also use int() and float() to convert between strings and integers. You will get an error if the string does not contain numeric characters.

The keyword “is” is a strong keyword in python generally used to compare the equality of True, False and None. We can use it only for strong comparison or else we can use “==” sign.

Different import modules in Python are string, re, datetime, math, random, os, multiprocessing, subprocess, socket, email, json, doctest, unittest, pdb, argparse and sys

Different exceptions are raised for different reasons.

Common exceptions:

ImportError: an import fails

IndexError: a list is indexed with an out-of-range number

NameError: an unknown variable is used

SyntaxError: the code can’t be parsed properly

TypeError: a function is called on a value of an inappropriate type

ValueError: a function is called on a value of the correct type, but with an inappropriate value

In except block, the raise statement can be used without arguments to re-raise whatever exception occurred.

An assertion is a sanity-check that you can turn on or off when you have finished testing the program. An expression is tested, and if the results comes up false, an exception is raised. Assertions are carried out through the use of the assert statement. Programmers often place assertion at the start of a function to check for a valid input, and after a function call to check for the valid output.

AssertionError exception can be caught and handled like any other exception using the try-except statement, but if not handled, this type of exception will terminate the entire program.

To determine whether a key is in a dictionary, you can use in and not in, just as you can for a list.

A useful dictionary method is get. It does the same thing as indexing, but if the key is not found in the dictionary it returns another specified value instead(‘None’, by default).

Tuples are very similar to lists, except that they are immutable(they cannot be changed). Also, they are created using parentheses, rather than square brackets.

Tuples are faster than lists, but they cannot be changed.

List comprehensions are a useful way of quickly creating lists whose contents obey a simple rule.

Ex: cubes = [i\*\*3 for I in range(5)]

print(cubes)

The output is [0, 1, 8, 27, 64]

A list comprehension can also contain an if statement to enforce a condition or values in the list.

Ex: evens = [i\*\*2 for i in range(10) if i\*\*2 % 2 == 0]

print(evens)

The output is [0, 4, 16, 36, 64]

Functional programming seeks to use pure functions. Pure functions have no side effects, and return a value that depends only on their arguments.

In Python finite generator can be converted to the lists by passing them as arguments to the list function.

Using generators results in improved performance, which is the result of the lazy generation of values, which translates to lower memory usage. Furthermore, we do not need to wait until all the elements have been generated before we start to use them.

There are two ways you can iterate through a loop using the strings:

Ex:

1. fruit = ‘banana’

for letter in fruit :

print(letter)

1. index = 0

while index < len(fruit) :

letter = fruit[index]

print(letter)

index = index + 1

In python if the slice operation refers to an index of a string which is out of range then it will still not give an out of index error. For s=”Python” the output s[2:12] will give the output “thon”

If we leave off the first number of the last number of the slice, it is assumed to be the beginning or end of the string respectively.

Opening a File:

Before we can read the contents of the file, we must tell python which file we are going to work with and what we will be doing with the file. This is done with the open() function. Open() returns a “file handle” - a variable used to perform operations on the file. Similar to “File -> Open” in a word processor.

Handle = open(filename, mode)

When files are missing: When a file is not present the FileNotFoundError occurs in python.

The newline character: We use a special character called the “newline” to indicate when a line ends. We represent it as \n in strings. Newline is still one character- not two

Ex:

Stuff = ‘X\nY’

Here in the above code the len(Stuff) will be 3 and not 4 or 2.

In python the print() statement always add the new line at the end of the statement.