Python CheatSheet

Basics

Basic syntax from the Python programming language

Showing Output To User

The print function is used to display or print output as follows:

print("Content that you wanna print on screen")

We can display the content present in an object using the print function as follows:

var1 = "Shruti"print("Hi my name is: ", var1)

Taking Input From the User

The input function is used to take input as a string or character from the user as follows:

var1 = input("Enter your name: ")print("My name is: ", var1)

To take input in the form of other data types, we need to typecast them as follows:

To take input as an integer:

var1 = int(input("Enter the integer value"))print(var1)

To take input as a float:

var1 = float(input("Enter the float value"))print(var1)

range **Function**

The range function returns a sequence of numbers, e.g., numbers starting from 0 to n-1 for range(0, n):

range(int_start_value, int_stop_value, int_step_value)

Here the start value and step value are by default 1 if not mentioned by the programmer,
but int_stop_value is a compulsory parameter in the range function.

Example:

Display all even numbers between 1 to 100:

for i in range(0, 101, 2): print(i)

Comments

Comments are used to make the code more understandable for programmers, and they are not executed by the compiler or interpreter.

Single Line Comment

This is a single line comment

Multi-line Comment

'''This is amulti-linecomment'''

Escape Sequence

An escape sequence is a sequence of characters that doesn't represent itself but is translated into another character when used inside a string literal or character. Some of the escape sequence characters are as follows:

Newline

Newline Character:

print("\n")

Backslash

It adds a backslash:

print("\\")

Single Quote

It adds a single quotation mark:

print("\'")

Tab

It gives a tab space:

print("\t")

Backspace

It adds a backspace:

print("\b")

Octal Value

It represents the value of an octal number:

print("\ooo")

Hex Value

It represents the value of a hex number:

print("\xhh")

Carriage Return

Carriage return or \r will just work as if you have shifted your cursor to the beginning of the string or line:

print("\r")

Strings

Python string is a sequence of characters, and each character can be individually accessed using its index.

String

You can create strings by enclosing text in both forms of quotes - single quotes or double quotes:

variable_name = "String Data"

Example:

str = "Shruti"print("string is ", str)

Indexing

The position of every character placed in the string starts from the 0th position and step by step it ends at length-1 position.

Slicing

Slicing refers to obtaining a sub-string from the given string. The following code will include index 1, 2, 3, and 4 for the variable named **var_name**.

Slicing of the string can be obtained by the following syntax:

string_var[int_start_value:int_stop_value:int_step_value]

var_name[1:5]

Here start and step value are considered 0 and 1 respectively if not mentioned by the programmer.

isalnum() **Method**

Returns True if all the characters in the string are alphanumeric, else False:

string_variable.isalnum()

isalpha() **Method**

Returns True if all the characters in the string are alphabets:

string_variable.isalpha()

isdecimal() **Method**

Returns True if all the characters in the string are decimals:

string_variable.isdecimal()

isdigit() **Method**

Returns True if all the characters in the string are digits:

string_variable.isdigit()

islower() **Method**

Returns True if all characters in the string are lower case:

string_variable.islower()

isspace() **Method**

Returns True if all characters in the string are whitespaces:

string_variable.isspace()

isupper() **Method**

Returns True if all characters in the string are upper case:

string_variable.isupper()

lower() **Method**

Converts a string into lower case equivalent:

string_variable.lower()

upper() **Method**

Converts a string into upper case equivalent:

string_variable.upper()

strip() **Method**

It removes leading and trailing spaces in the string:

string_variable.strip()

List

A list in Python represents a list of comma-separated values of any data type between square brackets:

var_name = [element1, element2, ...]

These elements can be of different data types.

Indexing

The position of every element placed in the list starts from the 0th position and step by step it ends at length-1 position. List is ordered, indexed, mutable, and the most flexible and dynamic collection of elements in Python.

Empty List

This method allows you to create an empty list:

my_list = []

index() **Method**

Returns the index of the first element with the specified value:

list.index(element)

append() **Method**

Adds an element at the end of the list:

list.append(element)

extend() **Method**

Add the elements of a given list (or any iterable) to the end of the current list:

list.extend(iterable)

insert() **Method**

Adds an element at the specified position:

list.insert(position, element)

pop() **Method**

Removes the element at the specified position and returns it:

list.pop(position)

remove() **Method**

The remove() method removes the first occurrence of a given item from the list:

list.remove(element)

clear() **Method**

Removes all the elements from the list:

list.clear()

count() **Method**

Returns the number of elements with the specified value:

list.count(value)

reverse() **Method**

Reverses the order of the list:

list.reverse()

sort() **Method**

Sorts the list:

list.sort(reverse=True|False)

Tuples

Tuples are represented as comma-separated values of any data type within parentheses.

Tuple Creation

variable_name = (element1, element2, ...)

These elements can be of different data types.

Indexing

The position of every element placed in the tuple starts from the 0th position and step by step it ends at length-1 position. Tuples are ordered, indexed, immutable, and the most secure collection of elements.

Let's talk about some of the tuple methods:

count() **Method**

It returns the number of times a specified value occurs in a tuple:

tuple.count(value)

index() **Method**

It searches the tuple for a specified value and returns the position:

tuple.index(value)

Sets

A set is a collection of multiple values which is both unordered and unindexed. It is written in curly brackets.

Set Creation: Way 1

var_name = {element1, element2, ...}

Set Creation: Way 2

var_name = set([element1, element2, ...])

Set is an unordered, immutable, non-indexed type of collection. Duplicate elements are not allowed in sets.

Set Methods

Let's talk about some of the methods of sets:

add() **Method**

Adds an element to a set:

set.add(element)

clear() **Method**

Remove all elements from a set:

set.clear()

discard() **Method**

Removes the specified item from the set:

set.discard(value)

intersection() **Method**

Returns the intersection of two or more sets:

set.intersection(set1, set2 ... etc)

issubset() **Method**

Checks if a set is a subset of another set:

set.issubset(set)

pop() **Method**

Removes an element from the set:

set.pop()

remove() **Method**

Removes the specified element from the set:

set.remove(item)

union() **Method**

Returns the union of two or more sets:

set.union(set1, set2...)

Dictionaries

The dictionary is an unordered set of comma-separated key:value pairs, within {}, with the requirement that within a dictionary, no two keys can be the same.

Dictionary

<dictionary-name> = {<key>: value, <key>: value ...}

Dictionary is an ordered and mutable collection of elements. Dictionary allows duplicate values but not duplicate keys.

Empty Dictionary

By putting two curly braces, you can create a blank dictionary:

mydict = {}

Adding Element to a Dictionary

By this method, one can add new elements to the dictionary:

<dictionary>[<key>] = <value>

Updating Element in a Dictionary

If a specified key already exists, then its value will get updated:

<dictionary>[<key>] = <value>

Deleting an Element from a Dictionary

del keyword is used to delete a specified key:value pair from the dictionary as follows:

del <dictionary>[<key>]

Dictionary Functions & Methods

Below are some of the methods of dictionaries:

len() **Method**

It returns the length of the dictionary, i.e., the count of elements (key: value pairs) in the dictionary:

len(dictionary)

clear() **Method**

Removes all the elements from the dictionary:

dictionary.clear()

get() **Method**

Returns the value of the specified key:

dictionary.get(keyname)

items() **Method**

Returns a list containing a tuple for each key-value pair:

dictionary.items()

keys() **Method**

Returns a list containing the dictionary's keys:

dictionary.keys()

values() **Method**

Returns a list of all the values in the dictionary:

dictionary.values()

update() **Method**

Updates the dictionary with the specified key-value pairs:

dictionary.update(iterable)

Indentation

In Python, indentation means the code is written with some spaces or tabs into many different blocks of code to indent it so that the interpreter can easily execute the Python code.

Indentation is applied on conditional statements and loop control statements. Indent specifies the block of code that is to be executed depending on the conditions.

Conditional Statements

The if, elif, and else statements are the conditional statements in Python, and these implement selection constructs (decision constructs).

if **Statement**

if (conditional expression): statements

if-else **Statement**

if (conditional expression): statementselse: statements

if-elif **Statement**

if (conditional expression): statementselif (conditional expression): statementselse: statements

Nested if-else **Statement**

if (conditional expression): if (conditional expression): statements else: statementselse: statements

Example:

a = 15b = 20c = 12if (a > b and a > c): print(a, "is greatest")elif (b > c and b > a): print(b, " is greatest")else: print(c, "is greatest")

Loops in Python

A loop or iteration statement repeatedly executes a statement, known as the loop body, until the controlling expression is false (0).

for **Loop**

The for loop of Python is designed to process the items of any sequence, such as a list or a string, one by one.

for <variable> in <sequence>: statements_to_repeat

Example:

for i in range(1, 101, 1): print(i)

while **Loop**

A while loop is a conditional loop that will repeat the instructions within itself as long as a conditional remains true.

while <logical-expression>: loop-body

Example:

i = 1while (i <= 100): print(i) i = i + 1

break **Statement**

The break statement enables a program to skip over a part of the code. A break statement terminates the very loop it lies within.

for <var> in <sequence>: statement1 if <condition>: break statement2statement_after_loop

Example:

for i in range(1, 101, 1): print(i, end=" ") if (i == 50): break else: print("Mississippi")print("Thank you")

continue **Statement**

The continue statement skips the rest of the loop statements and causes the next iteration to occur.

for <var> in <sequence>: statement1 if <condition>: continue statement2 statement3 statement4

Example:

for i in [2, 3, 4, 6, 8, 0]: if (i % 2 != 0): continue print(i)

Functions

A function is a block of code that performs a specific task. You can pass parameters into a function. It helps us to make our code more organized and manageable.

Function Definition

def my_function(): # statements

def keyword is used before defining the function.

Function Call

my_function()

Whenever we need that block of code in our program, simply call that function name whenever needed. If parameters are passed during defining the function, we have to pass the parameters while calling that function.

Example:

def add(): # function definition a = 10 b = 20 print(a + b)add() # function call

Return Statement in Python Function

The function return statement returns the specified value or data item to the caller.

return [value/expression]

Arguments in Python Function

Arguments are the values passed inside the parenthesis of the function while defining as well as while calling.

def my_function(arg1, arg2, arg3, ...argn): # statementsmy_function(arg1, arg2, arg3, ...argn)

Example:

def add(a, b): return a + bx = add(7, 8)print(x)

File Handling

File handling refers to reading or writing data from files. Python provides some functions that allow us to manipulate data in the files.

open() **Function**

var_name = open("file name", "mode")

Modes

1. **r** - to read the content from file
2. **w** - to write the content into file
3. **a** - to append the existing content into file
4. **r+** - To read and write data into the file. The previous data in the file will be overridden.
5. **w+** - To write and read data. It will override existing data.
6. **a+** - To append and read data from the file. It won't override existing data.

close() **Function**

var_name.close()

read() **Function**

The read functions contain different methods: read(), readline(), and readlines().

read() # return one big string

It returns a list of lines:

readlines() # returns a list

It returns one line at a time:

readline() # returns one line at a time

write() **Function**

This function writes a sequence of strings to the file.

write() # Used to write a fixed sequence of characters to a file

It is used to write a list of strings:

writelines()

Exception Handling

An exception is an unusual condition that results in an interruption in the flow of a program.

try **and** except

A basic try-catch block in Python. When the try block throws an error, the control goes to the except block.

try: [Statement body block] raise Exception()except Exceptionname: [Error processing block]

else

The else block is executed if the try block has not raised any exception and the code has been running successfully.

try: # statementsexcept: # statementselse: # statements

finally

The finally block will be executed even if the try block of code has been running successfully or the except block of code has been executed. The finally block of code will be executed compulsorily.

Object Oriented Programming (OOPS)

It is a programming approach that primarily focuses on using objects and classes. The objects can be any real-world entities.

class

The syntax for writing a class in Python:

class class_name: pass # statements

Creating an Object

Instantiating an object can be done as follows:

<object-name> =