**import sys**

**sys.path.append(r'C:\Users\julio.andrade\OneDrive - bbce.com.br\Autorregulação\ArquivosCVM')**

**import meu\_modulo**

**Python** is one of the most widely-used and popular programming languages, was developed by Guido van Rossum and released first in 1991. Python is a free and open-source language with a very simple and clean syntax which makes it easy for developers to **learn Python**. It supports object-oriented programming and is most commonly used to perform general-purpose programming. Python is used in several domains like Data Science, Machine Learning, Deep Learning, Artificial Intelligence, Scientific Computing Scripting, Networking, Game Development Web Development, Web Scraping, and various other domains.

To give a comprehensive overview of Python programming, we made a **Python Cheat Sheet** for Python programmers. In this Cheat Sheet of Python, you’ll learn all the basic to advanced topics and concepts of Python, like Python data types, Python for loop, Python slice, python map function, python dictionary, Python File Handling, etc.

**Data Type in Python**

The [type() function](https://www.geeksforgeeks.org/python-type-function/) can be used to define the values of various data types and to check their data types.

|  |
| --- |
| # DataType Output: str  x **=** "Hello World"    # DataType Output: int  x **=** 50    # DataType Output: float  x **=** 60.5    # DataType Output: complex  x **=** 3j    # DataType Output: list  x **=** ["geeks", "for", "geeks"]    # DataType Output: tuple  x **=** ("geeks", "for", "geeks")    # DataType Output: range  x **=** range(10)    # DataType Output: dict  x **=** {"name": "Suraj", "age": 24}    # DataType Output: set  x **=** {"geeks", "for", "geeks"}    # DataType Output: frozenset  x **=** frozenset({"geeks", "for", "geeks"})    # DataType Output: bool  x **=** True    # DataType Output: bytes  x **=** b"Geeks"    # DataType Output: bytearray  x **=** bytearray(4)    # DataType Output: memoryview  x **=** memoryview(bytes(6))    # DataType Output: NoneType  x **=** None |

**Python Program to Print Hello world**

The print() function in Python is used to print Python objects as strings as standard output.

|  |
| --- |
| # python program to print "Hello World"  print("Hello World") |

**Output:**

Hello World

**Python end parameter in print()**

The [keyword end](https://www.geeksforgeeks.org/gfact-50-python-end-parameter-in-print/) can be used to avoid the new line after the output or end the output with a different string.

|  |
| --- |
| # ends the output with a space  **print**("Welcome to", end**=**' ')  print("GeeksforGeeks", end**=**' ') |

**Output:**

Welcome to GeeksforGeeks

**Python sep parameter in print()**

The separator between the inputs to the print() method in Python is by default a space, however, this can be changed to any character, integer, or string of our choice. The [‘sep’ argument](https://www.geeksforgeeks.org/python-sep-parameter-print/) is used to do the same thing.

|  |
| --- |
| # code for disabling the softspace feature  **print**('09', '12', '2016', sep**=**'-')    # another example  print('Example', 'geeksforgeeks', sep**=**'@') |

**Output:**

09-12-2016  
Example@geeksforgeeks

**Python Input**

The [input()](https://www.geeksforgeeks.org/python-input-function/) method in Python is used to accept user input. By default, it returns the user input as a string. By default, the input() function accepts user input as a string.

|  |
| --- |
| # Python program showing  # a use of input()    val **=** input("Enter your value: ")  print(val) |

**Output:**

Enter your value: Hello Geeks  
Hello Geeks

**Python Comment**

[Comments in Python](https://www.geeksforgeeks.org/python-comments/) are the lines in the code that are ignored by the interpreter during the execution of the program. There are three types of comments in Python:

* Single line Comments
* Multiline Comments
* Docstring Comments

|  |
| --- |
| # Single Line comment    # Python program to demonstrate  # multiline comments    """ Python program to demonstrate   multiline comments"""    name **=** "geeksforgeeks"  print(name) |

**Output:**

Geeksforgeeks

**Operators in Python**

In general, [Operators](https://www.geeksforgeeks.org/python-operators/) are used to execute operations on values and variables. These are standard symbols used in logical and mathematical processes.

**Arithmetic Operators**

Python [Arithmetic Operators](https://www.geeksforgeeks.org/assignment-operators-in-python/) are used to perform mathematical operations like addition, subtraction, multiplication, and division.

|  |
| --- |
| # Examples of Arithmetic Operator  a **=** 9  b **=** 4    # Addition of numbers  add **=** a **+** b    # Subtraction of numbers  sub **=** a **-** b    # Multiplication of number  mul **=** a **\*** b    # Modulo of both number  mod **=** a **%** b    # Power  p **=** a **\*\*** b    # print results  print(add)  print(sub)  **print**(mul)  print(mod)  print(p) |

**Output:**

13  
5  
36  
1  
6561

**Comparison Operators**

When comparing values, relational operators are utilized. Depending on the criteria, it returns True or False. [Comparison Operators](https://www.geeksforgeeks.org/relational-operators-in-python/) are another name for these operators.

|  |
| --- |
| # Examples of Relational Operators  a **=** 13  b **=** 33    # a > b is False  **print**(a > b)    # a < b is True  **print**(a < b)    # a == b is False  print(a **==** b)    # a != b is True  **print**(a !**=** b)    # a >= b is False  **print**(a >**=** b)    # a <= b is True  **print**(a <**=** b) |

**Output:**

False  
True  
False  
True  
False  
True

**Logical Operators in Python**

[Logical operators](https://www.geeksforgeeks.org/python-logical-operators-with-examples-improvement-needed/) are used on conditional statements in Python (either True or False). They conduct the logical AND, OR, and NOT operations.

|  |
| --- |
| # Examples of Logical Operator  a **=** True  b **=** False    # Print a and b is False  **print**(a **and** b)    # Print a or b is True  **print**(a **or** b)    # Print not a is False  **print**(**not** a) |

**Output:**

False  
True  
False

**Bitwise Operators in Python**

[Bitwise operators](https://www.geeksforgeeks.org/python-bitwise-operators/) are used in Python to do bitwise operations on integers. After converting the numbers to binary, operations are done on each bit or corresponding pair of bits, hence the name bitwise operators.

|  |
| --- |
| # Examples of Bitwise operators  a **=** 10  b **=** 4    # Print bitwise AND operation  **print**(a & b)    # Print bitwise OR operation  **print**(a | b)    # Print bitwise NOT operation  print(~a)    # print bitwise XOR operation  print(a ^ b)    # print bitwise right shift operation  **print**(a >> 2)    # print bitwise left shift operation  print(a << 2) |

**Output:**

0  
14  
-11  
14  
2  
40

**String Slicing**

[Strings](https://www.geeksforgeeks.org/python-string/) in Python can be constructed with single, double, or even triple quotes. The slicing method is used to access a single character or a range of characters in a String. A Slicing operator (colon) is used to slice a String.

|  |
| --- |
| # Creating a String  String1 **=** "GeeksForGeeks"  **print**("Initial String: ")  **print**(String1)    # Printing 3rd character  print("\nSlicing characters from 3-12: ")  print(String1[3])    # Printing characters between  # 3rd and 2nd last character  print("\nSlicing characters between " **+**      "3rd and 2nd last character: ")  print(String1[3:**-**2]) |

**Output:**

Initial String:   
GeeksForGeeks  
Slicing characters from 3-12:   
k  
Slicing characters between 3rd and 2nd last character:   
ksForGee

**Conditional Statements**

Decision-making statements in programming languages decide the direction(Control Flow) of the flow of program execution.

**Python If-Else**

In a [conditional if Statement](https://www.geeksforgeeks.org/python3-if-if-else-nested-if-if-elif-statements/) the additional block of code is merged as an else statement which is performed when if condition is false.

|  |
| --- |
| # python program to illustrate If else statement    i **=** 20  **if** (i < 15):      print("i is smaller than 15")  **else**:      print("i is greater than 15")  print("i'm not in if and not in else Block") |

**Output:**

i is greater than 15  
i'm not in if and not in else Block

**Python For Loop**

The [Python For loop](https://www.geeksforgeeks.org/python-for-loops/) is used for sequential traversal, that is, iterating over an iterable such as a String, Tuple, List, Set, or Dictionary. For loops in Python only support collection-based iteration.

|  |
| --- |
| # Python program to illustrate  # Iterating over a list  l **=** ["geeks", "for", "geeks"]    **for** i **in** l:      print(i) |

**Output:**

geeks  
for  
geeks

**Python While Loop**

The [Python while Loop](https://www.geeksforgeeks.org/python-while-loop/) is used to execute a set of statements repeatedly until a condition is met. When the condition is met, the line immediately following the loop in the program is run.

|  |
| --- |
| # Python program to illustrate  # while loop  count **=** 0  **while** (count < 3):      count **=** count **+** 1      print("Hello Geek") |

**Output:**

Hello Geek  
Hello Geek  
Hello Geek

You can also read the use of [break, continue, and pass in Python](https://www.geeksforgeeks.org/break-continue-and-pass-in-python/).

**List in Python**

The [Python list](https://www.geeksforgeeks.org/python-lists/) is a sequence data type that is used to store the collection of data. Tuples and String are other types of sequence data types.

|  |
| --- |
| Var **=** ["Geeks", "for", "Geeks"]  print(Var) |

**Output:**

['Geeks', 'for', 'Geeks']

**List comprehension**

A Python [list comprehension](https://www.geeksforgeeks.org/python-list-comprehension/) is made up of brackets carrying the expression, which is run for each element, as well as the for loop, which is used to iterate over the Python list’s elements.

Also, Read – [Python Array](https://www.geeksforgeeks.org/python-arrays/)

|  |
| --- |
| # Using list comprehension to iterate through loop  List **=** [character **for** character **in** [1, 2, 3]]    # Displaying list  print(List) |

**Dictionary in Python**

A [dictionary in Python](https://www.geeksforgeeks.org/python-dictionary/) is a collection of key values, used to store data values like a map, which, unlike other data types holds only a single value as an element.

|  |
| --- |
| Dict **=** {1: 'Geeks', 2: 'For', 3: 'Geeks'}  print(Dict) |

**Output:**

{1: 'Geeks', 2: 'For', 3: 'Geeks'}

**Python Dictionary Comprehension**

Like List Comprehension, Python allows [dictionary comprehension](https://www.geeksforgeeks.org/python-dictionary-comprehension/). We can create dictionaries using simple expressions. A dictionary comprehension takes the form {key: value for (key, value) in iterable}

|  |
| --- |
| # Lists to represent keys and values  keys **=** ['a','b','c','d','e']  values **=** [1,2,3,4,5]    # but this line shows dict comprehension here  myDict **=** { k:v **for** (k,v) **in** zip(keys, values)}    # We can use below too  # myDict = dict(zip(keys, values))    print (myDict) |

**Output:**

{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}

**Tuples in Python**

[Tuple](https://www.geeksforgeeks.org/python-tuples/) is a list-like collection of Python objects. A tuple stores a succession of values of any kind, which are indexed by integers.

|  |
| --- |
| var **=** ("Geeks", "for", "Geeks")  print(var) |

**Output:**

('Geeks', 'for', 'Geeks')

**Sets in Python**

[Python Set](https://www.geeksforgeeks.org/python-sets/) is an unordered collection of data types that can be iterated, mutated and contains no duplicate elements. The order of the elements in a set is unknown, yet it may contain several elements.

|  |
| --- |
| var **=** {"Geeks", "for", "Geeks"}  print(var) |

**Output:**

{'for', 'Geeks'}

**Python Functions**

[Python Functions](https://www.geeksforgeeks.org/python-functions/) are a collection of statements that serve a specific purpose. The idea is to bring together some often or repeatedly performed actions and construct a function so that we can reuse the code included in it rather than writing the same code for different inputs over and over.

|  |
| --- |
| # A simple Python function  **def** fun():      print("Welcome to GFG")    # Driver code to call a function  fun() |

**Output:**

Welcome to GFG

**Function Arguments**

Arguments are the values given between the function’s parenthesis. A function can take as many parameters as it wants, separated by commas.

|  |
| --- |
| # A simple Python function to check  # whether x is even or odd  **def** evenOdd(x):  **if** (x **%** 2 **==** 0):          print("even")  **else**:          print("odd")      # Driver code to call the function  evenOdd(2)  evenOdd(3) |

**Output:**

even  
odd

**Return Statement in Python Function**

The function [return statement](https://www.geeksforgeeks.org/python-return-statement/) is used to terminate a function and return to the function caller with the provided value or data item.

|  |
| --- |
| # Python program to  # demonstrate return statement  **def** add(a, b):        # returning sum of a and b  **return** a **+** b    **def** is\_true(a):        # returning boolean of a  **return** bool(a)    # calling function  res **=** add(2, 3)  **print**("Result of add function is {}".format(res))    res **=** is\_true(2<5)  print("\nResult of is\_true function is {}".format(res)) |

**Output:**

Result of add function is 5  
Result of is\_true function is True

**The range() function**

The [Python range()](https://www.geeksforgeeks.org/python-range-function/) function returns a sequence of numbers, in a given range.

|  |
| --- |
| # print first 5 integers  # using python range() function  **for** i **in** range(5):      print(i, end**=**" ")  print() |

**Output:**

0 1 2 3 4

**Python Map Function**

The [map() function](https://www.geeksforgeeks.org/python-map-function/) returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable.

|  |
| --- |
| # Return double of n  **def** addition(n):  **return** n **+** n    # We double all numbers using map()  numbers **=** (1, 2, 3, 4)  result **=** map(addition, numbers)  print(list(result)) |

**Output:**

[2, 4, 6, 8]

**Python Filter Function**

The [filter() method](https://www.geeksforgeeks.org/filter-in-python/) filters the given sequence using a function that examines each element in the sequence to see if it is true or false.

|  |
| --- |
| # function that filters vowels  **def** fun(variable):      letters **=** ['a', 'e', 'i', 'o', 'u']  **if** (variable **in** letters):  **return** True  **else**:  **return** False      # sequence  sequence **=** ['g', 'e', 'e', 'j', 'k', 's', 'p', 'r']    # using filter function  filtered **=** filter(fun, sequence)    **print**('The filtered letters are:')  **for** s **in** filtered:  **print**(s) |

**Output:**

The filtered letters are:  
e  
e

**Python Reduce Function**

The [reduce function](https://www.geeksforgeeks.org/reduce-in-python/) is used to apply a certain function to all of the list components indicated in the sequence sent along.

|  |
| --- |
| **from** functools **import** reduce    nums **=** [1, 2, 3, 4]  ans **=** reduce(**lambda** x, y: x **+** y, nums)  print(ans) |

**Output:**

10

**Python Lambda**

Python [Lambda Functions](https://www.geeksforgeeks.org/python-lambda-anonymous-functions-filter-map-reduce/) are anonymous, which means they have no name. As we already know, the def keyword is used to define a normal function in Python. The lambda keyword in Python is used to declare an anonymous function.

|  |
| --- |
| calc **=** **lambda** num: "Even number" **if** num **%** 2 **==** 0 **else** "Odd number"    print(calc(20)) |

**Output:**

Even number

**\*args and \*\*kwargs in Python**

The [\*args and \*\*kwargs](https://www.geeksforgeeks.org/args-kwargs-python/) keywords allow functions to take variable-length parameters. The number of non-keyworded arguments and the action that can be performed on the tuple are specified by the \*args.\*\*kwargs, on the other hand, pass a variable number of keyword arguments dictionary to function, which can then do dictionary operations.

|  |
| --- |
| **def** myFun(arg1, arg2, arg3):  **print**("arg1:", arg1)  **print**("arg2:", arg2)  **print**("arg3:", arg3)      # Now we can use \*args or \*\*kwargs to  # pass arguments to this function :  args **=** ("Geeks", "for", "Geeks")  myFun(**\***args)    kwargs **=** {"arg1": "Geeks", "arg2": "for", "arg3": "Geeks"}  myFun(**\*\***kwargs) |

**Output:**

arg1: Geeks  
arg2: for  
arg3: Geeks  
arg1: Geeks  
arg2: for  
arg3: Geeks

**Try and Except Statement**

In Python, [Try and except statements](https://www.geeksforgeeks.org/python-try-except/) are used to catch and manage exceptions. Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside except clause.

|  |
| --- |
| a **=** [1, 2, 3]  **try**:  **print** ("Second element = %d" **%**(a[1]))        # Throws error since there are only 3 elements in array      print ("Fourth element = %d" **%**(a[3]))    **except**:  **print** ("An error occurred") |

**Output:**

Second element = 2  
An error occurred

**File Handling in Python**

Python too supports [file handling](https://www.geeksforgeeks.org/file-handling-python/) and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files.

|  |
| --- |
| **import** os    **def** create\_file(filename):  **try**:          with open(filename, 'w') as f:              f.write('Hello, world!\n')          print("File " **+** filename **+** " created successfully.")  **except** IOError:          print("Error: could not create file " **+** filename)    **def** read\_file(filename):  **try**:          with open(filename, 'r') as f:              contents **=** f.read()  **print**(contents)  **except** IOError:  **print**("Error: could not read file " **+** filename)    **def** append\_file(filename, text):  **try**:          with open(filename, 'a') as f:              f.write(text)  **print**("Text appended to file " **+** filename **+** " successfully.")  **except** IOError:          print("Error: could not append to file " **+** filename)    **def** rename\_file(filename, new\_filename):  **try**:          os.rename(filename, new\_filename)  **print**("File " **+** filename **+** " renamed to " **+**                    new\_filename **+** " successfully.")  **except** IOError:          print("Error: could not rename file " **+** filename)    **def** delete\_file(filename):  **try**:          os.remove(filename)  **print**("File " **+** filename **+** " deleted successfully.")  **except** IOError:  **print**("Error: could not delete file " **+** filename)      **if** \_\_name\_\_ **==** '\_\_main\_\_':      filename **=** "example.txt"      new\_filename **=** "new\_example.txt"        create\_file(filename)      read\_file(filename)      append\_file(filename, "This is some additional text.\n")      read\_file(filename)      rename\_file(filename, new\_filename)      read\_file(new\_filename)      delete\_file(new\_filename) |

**Python OOPs Concepts**

[Object-oriented Programming](https://www.geeksforgeeks.org/python-oops-concepts/) (OOPs) is a programming paradigm in Python that employs objects and classes. It seeks to include real-world entities such as inheritance, polymorphisms, encapsulation, and so on into programming. The primary idea behind OOPs is to join the data and the functions that act on it as a single unit so that no other portion of the code can access it.

* [Class And Objects](https://www.geeksforgeeks.org/python-classes-and-objects/)
* [Polymorphism](https://www.geeksforgeeks.org/polymorphism-in-python/)
* [Encapsulation](https://www.geeksforgeeks.org/encapsulation-in-python/)
* [Inheritance](https://www.geeksforgeeks.org/inheritance-in-python/)
* [Data Abstraction](https://www.geeksforgeeks.org/abstract-classes-in-python/)

In this example, we have a Car class with characteristics that represent the car’s make, model, and year. The \_make attribute is protected with a single underscore \_. The \_\_model attribute is marked as private with two underscores \_\_. The year attribute is open to the public.

We can use the getter function get\_make() to retrieve the protected attribute \_make. We can use the setter method set\_model() to edit the private attribute \_\_model. Using the getter method get\_model(), we may retrieve the changed private attribute \_\_model. There are no restrictions on accessing the public attribute year. We manage the visibility and accessibility of class members by using encapsulation with private and protected properties, offering a level of data hiding and abstraction.

|  |
| --- |
| **class** Car:  **def** \_\_init\_\_(self, make, model, year):          self.\_make **=** make  # protected attribute          self.\_\_model **=** model  # private attribute          self.year **=** year  # public attribute    **def** get\_make(self):  **return** self.\_make    **def** set\_model(self, model):          self.\_\_model **=** model    **def** get\_model(self):  **return** self.\_\_model      my\_car **=** Car("Toyota", "Corolla", 2022)    **print**(my\_car.get\_make())  # Accessing protected attribute  my\_car.set\_model("Camry")  # Modifying private attribute  **print**(my\_car.get\_model())  # Accessing modified private attribute  print(my\_car.year)  # Accessing public attribute |

**Output:**

Toyota  
Camry  
2022

**Python RegEx**

We define a pattern using a [regular expression](https://www.geeksforgeeks.org/python-regex/) to match email addresses. The pattern r”\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b” is a common pattern for matching email addresses. Using the re.search() function, the pattern is then found in the given text. If a match is found, we use the match object’s group() method to extract and print the matched email. Otherwise, a message indicating that no email was found is displayed.

|  |
| --- |
| **import** re  # Text to search  text **=** "Hello, my email is example@example.com"  # Define a pattern to match email addresses  pattern **=** r"\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b"  # Search for the pattern in the text  match **=** re.search(pattern, text)  # Check if a match is found  **if** match:  email **=** match.group()  print("Found email:", email)  **else**:  print("No email found.") |

**Output:**

Found email: [example@example.com](mailto:example@example.com)

**FAQs on Python**

**Q.1 What are the key features of Python?**

**Answer:**

*Features of Python are includes readability, simplicity, extensive library support, cross-platform compatibility, dynamic typing, a strong community, and object-oriented programming (OOP).*

**Q.2 What type of language is Python? Programming or scripting?**

**Answer:**

*Python is capable of scripting, but in a general sense, it is considered a general-purpose programming language.*

**Q.3 What is PEP 8?**

**Answer:**

*PEP stands for****Python Enhancement Proposal.****It is a set of rules that specify how to format Python code for maximum readability.*

**Q.4 What is the difference between .py and .pyc files?**

**Answer:**

*The .py files are the python source code files and the .pyc files contain the bytecode of the python files. .pyc files are generated when the code is imported from some other source. The interpreter converts the source .py files to .pyc files which helps by saving time.*