Python Cheat Sheet PDF: Download Updated Guide [2022]

This Python Cheat Sheet PDF Guide will help you to walk through the whole Python Programming Languages concept go through on one go.

Tutorials - Oct 18, 21 - by Susith Nonis - 10 min Read



List of content you will read in this article:

- 1. Python Cheat Sheet PDF
- 2. Conclusion

Python is an object-oriented, high-level programming language with dynamic semantics. It is an interpreted language. Its high-level data structures, along with dynamic typing and dynamic binding, make it highly appealing for rapid application creation, machine learning, and usage as a scripting language to link existing components. In this reference guide, we will share some quick Python Cheat Sheet PDF code that might be helpful for Python programmers to take some insightful help.

Python Cheat Sheet PDF

Basics of Python programming language

1. Basic Math Operations in Python

```
>>> 1 + 1  #addition
2

>>> 5 // 2     #integer division
```

```
2

>>> 5 / 2     #floating point division
2

>>> 5 * 2     #multiply two numbers
10

>>> 5 ** 2     #power of a number
2

>>> 5 % 2     #modulo operation
1
```

2. Print a Message on the Screen

```
>>> print("Hello")
Hello
```

3. Print ZEN of Python on screen

The Zen of Python is a collection of 19 "guiding principles" for developing computer programs that affect the Python programming language's architecture. Tim Peters, a software developer, wrote this collection of principles in 1999.

```
>>> import this
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

4. Assigning a variable with some value

```
>>> a = 2
>>> print(a)
2
>>> a
2

>>> a = "Hello"
>>> print(a)
Hello
>>> a
'Hello'
```

Data Types in Python

1. Assigning various types into variable

```
>>> a=2        #assigning an integer to a variable
>>> print(a)
2
>>> a
2

>>> a = "Hello Learner"  #assigning a string to a variable
>>> print(a)
Hello Learner
>>> a
'Hello Learner'

>>> a = 'a'  #assigning a character to a variable
>>> print(a)
a
>>> a
'a'
```

2. Printing types of data contained in a variable

```
>>> a = "Hello Learner"  #assigning a string to a variable
>>> type(a)
<class 'str'>
>>> a=2
>>> type(a)
<class 'int'>
```

Comments in Python

1. Single Line comments

```
# This is a comment
```

2. Docstrings

```
def fun():
    """
    This is a docstring
    """
```

Some commonly used functions

1. print()

```
>>> print('Hello!')
'Hello!'
```

2. len()

```
>>> len('Hey')
3
```

3. input()

```
>>> input('Enter some string')
Enter some string
```

Typecasting functions

```
>>> a='2'
>>> a=int(a)
>>> type(a)
```

```
<class 'int'>

>>> a=2
>>> a=str(a)
>>> type(a)
<class 'str'>
```

Flow control statements

```
>>> 1 == 1
True

>>> 2 != 1
True

>>> 1 == 1
True

>>> 3 > 1
True

>>> "one" == "two"
False
```

Boolean operations

```
>>> True and True
True

>>> True or False
False

>>> not True
False
```

Conditional statements

```
var = 'Python'
if var == 'Python':
    print('Hi, Python')
else:
    print('Hi, someone')

name = 'Python'
if name == 'Python':
    print('Hi, Python')
elif name == 'Java':
    print('You are Java')
else:
    print('You are neither Java nor Python.')
```

Loops in Python

```
idx = 0
while idx < 10:    #loop runs from 0 to 9
    print('Hello')
    idx = idx + 1

for i in range(0,4,2):    #start loop from 0, upto 4 with step size of 2
  print('Hello')
```

Loop control statements

1. Break statement

```
while True:
    print('What are you studying)
    inp = input()
    if inp == 'Nothing':
        break
print('The end')
```

2. Continue statement

```
while True:
    print('What are you studying)
    inp = input()
    if inp == 'Nothing':
        continue
    else:
    print("Good")
```

Functions

```
def fun():
    print('I am called')
fun()
```

```
def getValue(n):
    if n == 2:
        return 'It is 2'
    else:
        return 'It is not 2'
what = getValue(2)
print(what)
```

Lists in Python

One of the most used data structures in Python is a list, which is an ordered and mutable Python container. To make a list, put the components inside square brackets [] and separate them with commas.

1. Creating arrays

```
>>> arr = ['A', 'B']
>>> arr
['A', 'B']
```

2. Indexing arrays

```
>>> arr = ['A', 'B']
>>> arr[0]
'A'
```

```
>>> arr = ['A', 'B']
>>> arr[-1]
'B'
```

3. List slicing

```
>>> arr = ['A', 'B', 'C']
>>> arr[0:2]
['A', 'B']
```

4. Appending to a list

```
>>> arr = ['A', 'B']

>>> arr.remove('B')
```

5. Remove an element from the list

```
>>> arr = ['A', 'B']
>>> arr.remove('B')
```

6. Sorting a list

```
>>> arr = [2, 1]
>>> arr.sort()
>>> arr
[1, 2]
```

Strings in Python

Strings in Python are arrays of bytes that represent Unicode characters. Python lacks character data types. Most of the functions that work with lists can also be used with strings in Python.

1. Indexing

```
>>> string = 'Hello!'
>>> string[0]
'H'
```

2. upper() and lower() methods

```
>>> string = 'Hello!'
>>> string = spam.upper()
>>> string
'HELLO!'

>>> string = "HEY"
>>> string = string.lower()
>>> string
'hey'
```

3. join() and split()

```
>>> ', '.join(['A', 'B', 'C'])
'A, B, C'

>>> 'HeyKMan'.split('K')
['Hey', 'Man']
```

4. String Manipulation

```
>>> print("Hello!\nHow are you?\nI\'m great")
Hello!
How are you?
I'm great
```

Tuple in Python

Tuples are a data structure in Python that stores an ordered series of values. Tuples are immutable. This indicates that the values of a tuple cannot be changed.

```
a = ('A', 1)
>>> a[0]
'A'
```

```
>>> tuple(['A', 'B'])
('A', 'B')
```

Dictionaries in Python

In Python, a dictionary is an unordered collection of data values that are used to store data values similar to a map. Unlike other Data Types that contain only a single value as an element, a dictionary maintains a key, value pair.

1. Creating a dictionary

```
hash = {'key1': 'value1', 'key2': 'value2'}
```

2. Indexing in dictionaries

```
>>> hash = {'key1': 'value1', 'key2': 'value2'}
>>> hash['key1']
'value1'
```

3. Iterating dictionaries

```
>>> hash = {'key1': 'value1', 'key2': 'value2'}
>>>
>>> for k, v in hash.items():
>>>     print('Key: {} Value: {}'.format(k, str(v)))
Key: key2 Value: value2
Key: key1 Value: value1
```

Sets in Python

A set is an unordered collection of items with no duplicates. Set can be used to eliminate duplicate entries. Set objects are also capable of performing mathematical operations such as union, intersection, difference, and symmetric difference. The sets cannot be indexed.

1. Initializing sets

```
>>> s = {1, 2, 3, 2, 3, 4}
>>> s
{1, 2, 3, 4}
```

2. Adding to sets

```
>>> s = {1, 2, 3}
>>> s.add(4)
>>> s
{1, 2, 3, 4}
```

3. Removing from sets

```
>>> s = {1, 2, 3}
>>> s.remove(3)
>>> s
{1, 2}
```

4. Sets Union

union() will create a new set that contains all the elements from the sets provided.

```
>>> s1 = {1, 2, 3}
>>> s2 = {3, 4, 5}
>>> s1.union(s2)
{1, 2, 3, 4, 5}
```

5. Sets Intersection

The intersection will return a set containing only the elements that are common to all of them.

```
>>> s1 = {1, 2, 3}
>>> s2 = {2, 3, 4}
>>> s3 = {3, 4, 5}
>>> s1.intersection(s2, s3)
{3}
```

6. Lambda Functions

```
>>> sum = lambda x, y: x + y
>>> sum(1, 2)
3
```

Comprehensions

1. List comprehension

```
>>> a = [1, 2, 3]
>>> [i for i in a]
[1, 2, 3]
```

2. Sets comprehension

```
>>> Set = {"a", "b"}
>>> {s.upper() for s in Set}
{"A", "B"}
```

3. Exception Handling

```
try:
        result = x // y
        print("Answer is :", result)
    except ZeroDivisionError:
        print("Cannot divide by zero")

try:
    k = 5//0
except ZeroDivisionError:
    print("Cannot divide by zero")
finally:
    print('This is always executed')
```

Python modules

A module is a Python object with arbitrarily named attributes that you can import and use in your codes. Simply speaking, a module is a file consisting of some useful Python code.

You can import a module using the below syntax in your script.

```
import module_name
```

Below are some useful modules that can be helpful to you while coding.

- itertools
- re (Regular Expression)
- logging
- functools

- math

Conclusion
In this complete tutorial, We have prepared a list of necessary Python Cheat Sheet PDF reference guide for you from beginners to advanced levels. While various Python developers are looking for this kind of cheat sheet because remembering each code is too tough and very time-consuming for developers, So they want some sort of information like this cheat sheet.

This python cheat sheet PDF you can also download from the above-given download button is helpful for both types of programmers i.e. developers and data scientists, as this guide provides many of the quick Python solutions without keeping them into your memory while doing Python programming or creating applications.

People Are Also Reading:

- How to Learn Python?
- How to Update Python Version?
- How to Install Python on Windows?
- Update PIP

MonoVM Services: Dedicated Server, Domain Services, VPS Hosting, Web Hosting and SSL Certificate

Follow us on social media: