Python Cheatsheet

Table of Content:

Python Basics

Math Operators

From **Higuest** to **Lowest** precedence:

Operators	Operation	Example
**	Exponent	2 ** 3 = 8
%	Modulus/Remaider	22 % 8 = 16
//	Integer division	22 // 8 = 2
/	Division	22 / 8 = 2.75
*	Multiplication	3 * 3 = 15
-	Subtraction	5 - 2 = 3
+	Addition	2 + 2 = 4

Examples of expressions in the interactive shell:

```
>>> 2 + 3 * 6
20

>>> (2 + 3) * 6
30

>>> 2 ** 8
256

>>> 23 // 7
3

>>> 23 % 7
2

>>> (5 - 1) * ((7 + 1) / (3 - 1))
```

```
16.0
```

Data Types

Data Type	Examples
Integers	-2, -1, 0, 1, 2, 3, 4, 5
Floating-point numbers	-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25
Strings	'a', 'aa', 'aaa', 'Hello!', '11 cats'

String Concatenation and Replication

String concatenation:

```
>>> 'Alice' + 'Bob'
'AliceBob'
```

String Replication:

```
>>> 'Alice' * 5
'AliceAliceAliceAliceAlice'
```

Variables

You can name a variable anything as long as it obeys the following three rules:

1. It can be only one word.
2. It can use only letters, numbers, and the underscore (_) character.
3. It can't begin with a number.

Example:

```
>>> spam = 'Hello'
>>> spam
'Hello'
```

Comments

Inline comment:

```
# This is a comment
```

Multiline Comment:

```
"""
This is a Multiline Comment
You can also use:
''' multiline comment '''
"""
```

The print() Function

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

The input() Function

Example Code:

```
>>> print('What is your name?')    # ask for their name
>>> myName = input()
>>> print('It is good to meet you, ' + myName)
```

Output:

```
What is your name?
Al
It is good to meet you, Al
```

The len() Function

Evaluates to the integer value of the number of characters in a string:

```
>>> len('hello')
5
```

The str(), int(), and float() Functions

Convert Between Data Types:

Integer to String or Float:

```
>>> str(29)
'29'

>>> print('I am ' + str(29) + ' years old.')
I am 29 years old.

>>> str(-3.14)
'-3.14'
```

Float to Integer:

```
>>> int(7.7)
7

>>> int(7.7) + 1
8
```

Flow Control

Comparison Operators

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater Than
<=	Less than or Equal to
>=	Greater than or Equal to

These operators evaluate to True or False depending on the values you give them:

Examples:

```
>>> 42 == 42
True

>>> 40 == 42
False

>>> 'hello' == 'hello'
True
```

```
>>> 'hello' == 'Hello'
False

>>> 'dog' != 'cat'
True

>>> True == True
True

>>> True != False
True

>>> 42 == 42.0
True

>>> 42 == '42'
False
```

Boolean Operators

There are three Boolean operators: and, or, and not.

The *and* Operator's *Truth* Table:

Expression	Evaluates to
True and True	True
True and False	False
False and True	False
False and False	False

The *or* Operator's *Truth* Table:

Expression	Evaluates to
True or True	True
True or False	True
False or True	True
False or False	False

The *not* Operator's *Truth* Table:

Expression	Evaluates to
not True	False

not False True

Mixing Boolean and Comparison Operators

```
>>> (4 < 5) and (5 < 6)
True

>>> (4 < 5) and (9 < 6)
False

>>> (1 == 2) or (2 == 2)
True
```

You can also use multiple Boolean operators in an expression, along with the comparison operators:

```
>>> 2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
True
```

if Statements

```
if name == 'Alice':
    print('Hi, Alice.')
```

else Statements

```
name = 'Bob'
if name == 'Alice':
    print('Hi, Alice.')
else:
    print('Hello, stranger.')
```

elif Statements

```
name = 'Bob'
age = 5
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
```

```
name = 'Bob'
age = 30
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
else:
    print('You are neither Alice nor a little kid.')
```

while Loop Statements

```
spam = 0
while spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

break Statements

If the execution reaches a break statement, it immediately exits the while loop's clause.

```
while True:
    print('Please type your name.')
    name = input()
    if name == 'your name':
        break
print('Thank you!')
```

continue Statements

When the program execution reaches a continue statement, the program execution immediately jumps back to the start of the loop.

```
while True:
  print('Who are you?')
  name = input()
  if name != 'Joe':       #(1)
    continue              #(2)
  print('Hello, Joe. What is the password? (It is a fish.)')
  password = input()      #(3)
  if password == 'swordfish':
    break                 #(4)
print('Access granted.')  #(5)
```

for Loops and the range() Function

```
print('My name is')
for i in range(5):
    print('Jimmy Five Times (' + str(i) + ')')
```

Output:

My name is Jimmy Five Times (0) Jimmy Five Times (1) Jimmy Five Times (2) Jimmy Five Times (3) Jimmy Five Times (4)

The *range()* function can also be called with three arguments. The first two arguments will be the start and stop values, and the third will be the step argument. The step is the amount that the variable is increased by after each iteration.

```
for i in range(0, 10, 2):
    print(i)
```

Output:

```
0
2
4
6
8
```

You can even use a negative number for the step argument to make the for loop count down instead of up.

```
for i in range(5, -1, -1):
    print(i)
```

Output:

```
5
4
3
2
1
0
```

Importing Modules

```
import random
for i in range(5):
    print(random.randint(1, 10))
```

```
import random, sys, os, math
```

```
from random import *.
```

Ending a Program Early with sys.exit()

```
import sys

while True:
    print('Type exit to exit.')
    response = input()
    if response == 'exit':
        sys.exit()
    print('You typed ' + response + '.')
```

Functions

```
def hello(name):
    print('Hello ' + name)

hello('Alice')
hello('Bob')
```

Output:

```
Hello Alice
Hello Bob
```

Return Values and return Statements

When creating a function using the def statement, you can specify what the return value should be with a return statement. A return statement consists of the following:

- The return keyword.

- The value or expression that the function should return.

```
import random
def getAnswer(answerNumber):
    if answerNumber == 1:
        return 'It is certain'
    elif answerNumber == 2:
        return 'It is decidedly so'
    elif answerNumber == 3:
        return 'Yes'
    elif answerNumber == 4:
        return 'Reply hazy try again'
    elif answerNumber == 5:
        return 'Ask again later'
    elif answerNumber == 6:
        return 'Concentrate and ask again'
    elif answerNumber == 7:
        return 'My reply is no'
    elif answerNumber == 8:
        return 'Outlook not so good'
    elif answerNumber == 9:
        return 'Very doubtful'

r = random.randint(1, 9)
fortune = getAnswer(r)
print(fortune)
```

The None Value

```
>>> spam = print('Hello!')
Hello!
>>> None == spam
True
```

Keyword Arguments and print()

```
print('Hello', end='')
print('World')
```

Output:

```
HelloWorld
```

```
>>> print('cats', 'dogs', 'mice')
cats dogs mice
```

```
>>> print('cats', 'dogs', 'mice', sep=',')
cats,dogs,mice
```

Local and Global Scope

- Code in the global scope cannot use any local variables.
- However, a local scope can access global variables.
- Code in a function's local scope cannot use variables in any other local scope.
- You can use the same name for different variables if they are in different scopes. That is, there can be a local variable named spam and a global variable also named spam.

The global Statement

If you need to modify a global variable from within a function, use the global statement:

```
def spam():
    global eggs
    eggs = 'spam'

eggs = 'global'
spam()
print(eggs)
```

Output:

```
spam
```

There are four rules to tell whether a variable is in a local scope or global scope:

1. If a variable is being used in the global scope (that is, outside of all functions), then it is always a global variable.
2. If there is a global statement for that variable in a function, it is a global variable.
3. Otherwise, if the variable is used in an assignment statement in the function, it is a local variable.
4. But if the variable is not used in an assignment statement, it is a global variable.

Exception Handling

```
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print('Error: Invalid argument.')

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Output:

```
21.0
3.5
Error: Invalid argument.
None
42.0
```

Dictionaries and Structuring Data

Example Dictionary:

```
myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud'}
```

The keys(), values(), and items() Methods

values():

```
>>> spam = {'color': 'red', 'age': 42}
>>> for v in spam.values():
        print(v)
```

Output:

```
red
42
```

keys():

```
>>> for k in spam.keys():
        print(k)
```

Output:

```
color
age
```

items():

```
>>> for i in spam.items():
        print(i)
```

Output:

```
('color', 'red')
('age', 42)
```

Using the keys(), values(), and items() methods, a for loop can iterate over the keys, values, or key-value pairs in a dictionary, respectively

```
>>> spam = {'color': 'red', 'age': 42}
>>> for k, v in spam.items():
        print('Key: ' + k + ' Value: ' + str(v))
```

Output:

```
Key: age Value: 42
Key: color Value: red
```

Checking Whether a Key or Value Exists in a Dictionary

```
>>> spam = {'name': 'Zophie', 'age': 7}
```

```
>>> 'name' in spam.keys()
True

>>> 'Zophie' in spam.values()
True

>>> 'color' in spam.keys()
False

>>> 'color' not in spam.keys()
True

>>> 'color' in spam
False
```

The get() Method

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems.get('cups', 0)) + ' cups.'
'I am bringing 2 cups.'
>>> 'I am bringing ' + str(picnicItems.get('eggs', 0)) + ' eggs.'
'I am bringing 0 eggs.'
```

The setdefault() Method

```
spam = {'name': 'Pooka', 'age': 5}
if 'color' not in spam:
    spam['color'] = 'black'
```

The above code is equal to:

```
>>> spam = {'name': 'Pooka', 'age': 5}
>>> spam.setdefault('color', 'black')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
>>> spam.setdefault('color', 'white')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
```

Pretty Printing

```
import pprint
message = 'It was a bright cold day in April, and the clocks were striking
thirteen.'
count = {}

for character in message:
    count.setdefault(character, 0)
    count[character] = count[character] + 1

pprint.pprint(count)
```

Output:

```
{' ': 13,
 ',': 1,
 '.': 1,
 'A': 1,
 'I': 1,
 'a': 4,
 'b': 1,
 'c': 3,
 'd': 3,
 'e': 5,
 'g': 2,
 'h': 3,
 'i': 6,
 'k': 2,
 'l': 3,
 'n': 4,
 'o': 2,
 'p': 1,
 'r': 5,
 's': 3,
 't': 6,
 'w': 2,
 'y': 1}
```

Manipulating Strings

Escape Characters

Escape character	Prints as
'	Single quote
"	Double quote
\t	Tab

\n	Newline (line break)
\	Backslash

Example:

```
>>> print("Hello there!\nHow are you?\nI\'m doing fine.")
```

Output:

```
Hello there!
How are you?
I'm doing fine.
```

Raw Strings

A raw string completely ignores all escape characters and prints any backslash that appears in the string.

```
>>> print(r'That is Carol\'s cat.')
```

Output:

```
That is Carol\'s cat.
```

Multiline Strings with Triple Quotes

```
print('''Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and extortion.

Sincerely,
Bob''')
```

Output:

```
Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and extortion.
```

```
Sincerely,
Bob
```

Indexing and Slicing Strings

```
H   e   l   l   o       w   o   r   l   d   !
0   1   2   3   4   5   6   7   8   9   10  11
```

```
>>> spam = 'Hello world!'

>>> spam[0]
'H'

>>> spam[4]
'o'

>>> spam[-1]
'!'

>>> spam[0:5]
'Hello'

>>> spam[:5]
'Hello'

>>> spam[6:]
'world!'
```

Slicing:

```
>>> spam = 'Hello world!'

>>> fizz = spam[0:5]

>>> fizz
'Hello'
```

The in and not in Operators with Strings

```
>>> 'Hello' in 'Hello World'
True

>>> 'Hello' in 'Hello'
```

```
True

>>> 'HELLO' in 'Hello World'
False

>>> '' in 'spam'
True

>>> 'cats' not in 'cats and dogs'
False
```

The upper(), lower(), isupper(), and islower() String Methods

upper() and lower():

```
>>> spam = 'Hello world!'

>>> spam = spam.upper()

>>> spam
'HELLO WORLD!'

>>> spam = spam.lower()

>>> spam
'hello world!'
```

isupper() and islower():

```
>>> spam = 'Hello world!'

>>> spam.islower()
False

>>> spam.isupper()
False

>>> 'HELLO'.isupper()
True

>>> 'abc12345'.islower()
True

>>> '12345'.islower()
False

>>> '12345'.isupper()
False
```

The isX String Methods

- **isalpha()** returns True if the string consists only of letters and is not blank.
- **isalnum()** returns True if the string consists only of lettersand numbers and is not blank.
- **isdecimal()** returns True if the string consists only ofnumeric characters and is not blank.
- **isspace()** returns True if the string consists only of spaces,tabs, and new-lines and is not blank.
- **istitle()** returns True if the string consists only of wordsthat begin with an uppercase letter followed by onlylowercase letters.

The startswith() and endswith() String Methods

```
>>> 'Hello world!'.startswith('Hello')
True

>>> 'Hello world!'.endswith('world!')
True

>>> 'abc123'.startswith('abcdef')
False

>>> 'abc123'.endswith('12')
False

>>> 'Hello world!'.startswith('Hello world!')
True

>>> 'Hello world!'.endswith('Hello world!')
True
```

The join() and split() String Methods

join():

```
>>> ', '.join(['cats', 'rats', 'bats'])
'cats, rats, bats'

>>> ' '.join(['My', 'name', 'is', 'Simon'])
'My name is Simon'

>>> 'ABC'.join(['My', 'name', 'is', 'Simon'])
'MyABCnameABCisABCSimon'
```

split():

```
>>> 'My name is Simon'.split()
['My', 'name', 'is', 'Simon']

>>> 'MyABCnameABCisABCSimon'.split('ABC')
['My', 'name', 'is', 'Simon']

>>> 'My name is Simon'.split('m')
['My na', 'e is Si', 'on']
```

Justifying Text with rjust(), ljust(), and center()

rjust() and ljust():

```
>>> 'Hello'.rjust(10)
'     Hello'

>>> 'Hello'.rjust(20)
'               Hello'

>>> 'Hello World'.rjust(20)
'         Hello World'

>>> 'Hello'.ljust(10)
'Hello     '
```

An optional second argument to rjust() and ljust() will specify a fill character other than a space character. Enter the following into the interactive shell:

```
>>> 'Hello'.rjust(20, '*')
'***************Hello'

>>> 'Hello'.ljust(20, '-')
'Hello---------------'
```

center():

```
>>> 'Hello'.center(20)
'       Hello        '

>>> 'Hello'.center(20, '=')
'=======Hello========'
```

Removing Whitespace with strip(), rstrip(), and lstrip()

```
>>> spam = '    Hello World     '

>>> spam.strip()
'Hello World'

>>> spam.lstrip()
'Hello World '

>>> spam.rstrip()
'    Hello World'
```

```
>>> spam = 'SpamSpamBaconSpamEggsSpamSpam'
>>> spam.strip('ampS')

'BaconSpamEggs'
```

Copying and Pasting Strings with the pyperclip Module

```
>>> import pyperclip

>>> pyperclip.copy('Hello world!')

>>> pyperclip.paste()
'Hello world!'
```