Python Basics Cheat Sheet - Hackr.io

Math Operators

You can perform math operations like addition, subtraction, multiplication, and division using arithmetic operators in Python. You can also access several libraries that can help you with more advanced arithmetic problems. Here's a quick list of some operators and their functions:

**
Find **exponents**

%
Find the **remainder**.

//
Perform **Integer division**.

/
Perform **Division** operations.

*
Perform **Multiplication** operations.

-
Perform **Subtraction** operations.

+
Perform **Addition** operations.

Examples

```
 >>> 3 * 8 + 6 + 0
30

>>> (2 + 3) * 6
30
```

```
>>> 5 ** 6
15625
```

Data Types

A data type is a mechanism to inform the compiler which data (integer, character, float, etc.) should be stored and how much memory to allocate as a result.

Here are Python's data types:

1. Numbers (float, complex or floating-point)
2. Sequence (strings, list, tuples, etc.)
3. Boolean (True or False)
4. Set
5. Dictionary

```
>>> a = 5.5 # float datatype
>>> a
5.5

>>> a = 5 # int datatype
>>> a
5

>>> a = [1, 2, 3, 4, 5, 6] # list datatype
>>> a
[1, 2, 3, 4, 5, 6]

>>> a = 'hello' # string datatype
>>> a
'hello'


>>> a = {1, 2, 3} # set datatype
>>> a
{1, 2, 3}


>>> a = True # boolean datatype
>>> a
```

```
True

>>> a = {1: 2} # dictionary datatype
>>> a
{1: 2}
```

Variables

A variable is a memory area where data is kept in any programming language. This area is usually present inside the RAM at a given address. Variables may hold any value, including numbers, text, and true/false values. As a result, if you wish to use that value at any point in the program, you may simply use the variable that has that value.

It's worth noting that because Python isn't a highly typed language, you don't have to specify the type of variable based on the value it holds. The type of data stored in a variable will be decoded implicitly at run time in Python, and determined by the type of data stored in that variable.

```
>>> a = 'This is a string variable'
>>> a
'This is a string variable'
```

```
>>> a = 5
>>> a
5
```

Comments

A good programming practice is to leave comments for yourself and others, regardless of the programming language. While python is simpler to understand than Java, c++, and other languages, it’s only polite to leave comments to offer clarification on the file’s purpose.

Inline Comment

```
# This is an inline comment
```

Multiline Comment

```
"""
```

```
This is a
multiline comment

"""
```

Printing Output

The **print**() method sends a message to the screen or another standard output device. The message can be a string or another object, which will be converted to a string before being displayed on the screen.

```
>>> print('How are you?')
How are you?
>>> x = 10
>>> print('Hello world!', x)
Hello world! 10
```

input()

When the **input**() function is called, the program execution is halted until the user provides an input.

```
The input() Function
>>> print('How are you?')
>>> myStatus = input()
>>> print('Nice to meet you, {}'.format(myStatus))
How are you?
Al
Nice to meet you, Al
```

Len() Function

The **len**() function returns the number of elements in a sequential or a random data structure like list, string, set.

```
>>> len('Computer')
8
```

Typecasting Functions

Here's how to convert integers to float or string:

```
>>> str(14)
'14'

>>> print('He is {} years old'.format(str(14)))
He is 14 years old.
>>> str(-4.89)
'-4.89'
```

Here's how to convert float to integer:

```
>>> int(6.7)
6
>>> int(6.6) + 1
7
```

Flow Control

Comparison Operators

==
Equal to

!=
Not equal to

<
Less than

>
Greater Than

<=
Less than or Equal to

>=

Greater than or Equal to

```
>>> 71 == 70
False
>>> 40 == 34
False
>>> 'man' == 'man'
True
>>> 'man' == 'Man'
False
>>> 'bat' != 'butterfly'
True
>>> 50 == 50.0
True
>>> 1 == '1'
False
```

Boolean Evaluation

```
>>> True == True
True

>>> True != False
True


>>> True is True
True
>>> True is not False
True

>>> if a is True:
>>>     pass
>>> if a is not False:
>>>     pass
>>> if a:
>>>     pass

>>> if a is False:
```

```
>>>    pass
>>> if a is not True:
>>>    pass
>>> if not a:
>>>    pass
```

Boolean Operators

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

Here's the truth table for the "**and**" operator:

True and True	True
True and False	False
False and True	False
False and False	False

Here's the truth table for the "**not**" operator

not True	False
not False	True

Finally, here's the truth table for "**or**" operator

True or True	True
True or False	True
False or True	True
False or False	False

Mixing Boolean and Comparison Operators

```
>>> (43< 57) and (3 < 9)
True
>>> (14 < 15) and (92< 61)
False
>>> (1 == 3) or (4 == 4)
True
```

In addition to the comparison operators, you can use several Boolean operators in an expression:

```
>>> 2 + 2 == 4 and not 2 + 2 == 6 and 2 * 2 == 2 + 2
True
```

If-Else Statements

```
name = 'Peter'
if name == 'Peter':
    print('Hello, Peter')
```

Output
Hello, Peter

```
name = 'Mike'
if name == 'Peter':
    print('Hello, Peter.')
else:
    print('Hello, anonymous')
```

Output
Hello, anonymous

Combining If and Else (elif statement)

```
name = 'Mike'
age = 5
if name == 'Peter':
    print('Hi, Peter.')
elif age < 10:
    print('Your age is less than 10')

name = 'Mike'
age = 30
if name == 'Peter':
```

```
    print('Hello, Peter.')
elif age < 10:
    print('Your age is less than 12')
else:
    print('Your age is more than 10')
```

Output
Your age is less than 10
Your age is more than 10

While Loop Statements

While loop statements are used to run a block of code for a specified number of times:

```
var = 0
while var < 10:
    print('Hello, world.')
    var = var + 1
```

Output
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.

Break Statement

If the execution reaches a **break** statement, the iteration is stopped and the control exits from the loop.

```
var = 1
```

```
while True:
    print('This block of code is running...')
    if var == 10:
        break
    var += 1

print('Loop exited')
```

Output
This block of code is running...
This block of code is running...
This block of code is running...
This block of code is running...
This block of code is running...
This block of code is running...
This block of code is running...
This block of code is running...
This block of code is running...
This block of code is running...
Loop exited

Continue Statement

The control restarts from the beginning of the loop once the program encounters the **continue** statement.

```
var = 0

while var <= 10:

    var += 1

    if var == 5:
        continue

    print('This block of code is running for number... ', var)

print('Loop exited')
```

Output

```
This block of code is running for number...  1
This block of code is running for number...  2
This block of code is running for number...  3
This block of code is running for number...  4
This block of code is running for number...  6
This block of code is running for number...  7
This block of code is running for number...  8
This block of code is running for number...  9
This block of code is running for number...  10
This block of code is running for number...  11
Loop exited
```

For Loop

A **for** loop is controlled by a sequence, such as an iterator, a list, or another collection type. The body of the loop is run for each item in the series, and the loop finishes when the sequence is exhausted.

```
for var in range(1, 10):

    print("Loop running...")

print('Loop exited')
```

Output

```
Loop running...
Loop running...
Loop running...
Loop running...
Loop running...
Loop running...
Loop running...
Loop running...
Loop running...
Loop exited
```

Range Function

Programmers use Python's **range**() to run an iteration for a specified number of times. It takes the following arguments:

Start: the number that the sequence of integers should begin with.
Stop: the integer before which the integer sequence should be returned. Stop – 1 is the end of the integer range. Stop – 1 is the end of the integer range.
Step: the integer value that determines the increase between each integer in the sequence.

```
for var in range(1, 20, 2):

    print("Loop running with step size of 2...")

print('Loop exited')
```

Output
Loop running with step size of 2...
Loop running with step size of 2...
Loop running with step size of 2...
Loop running with step size of 2...
Loop running with step size of 2...
Loop running with step size of 2...
Loop running with step size of 2...
Loop running with step size of 2...
Loop running with step size of 2...
Loop running with step size of 2...
Loop exited

For-If- Else Statements Combined

For-if-else statements allow you to provide conditional statements inside the loops including the **if**, **else** and **elif**.

```
for var in range(1, 11):

    if(var%2==0):
        print("This is even integer")
    else:
        print("This is odd integer")

print('Loop exited')
```

Output
This is odd integer
This is even integer
This is odd integer
This is even integer
This is odd integer
This is even integer
This is odd integer
This is even integer
This is odd integer
This is even integer
Loop exited

Modules in Python

We can import other python module codes by importing file/function from other python modules using the **import** statement of Python. The **import** statement is the most frequent method of triggering the import mechanism, but it isn't the only means for import.

```
import random
for i in range(5):
    print("Random integer is", random.randint(1, 30))
```

Output
Random integer is 8
Random integer is 10
Random integer is 11
Random integer is 3
Random integer is 8

We can also use the **from** statement to import a specified method of the module

```
from collections import Counter

List = [1, 2, 3, 4, 5, 5, 1]
Cnt = Counter(List)
```

```
print(Cnt)
```

Output
Counter({1: 2, 5: 2, 2: 1, 3: 1, 4: 1})

Function

A function is a reusable, ordered block of code that performs a single, connected activity. Functions provide your program more modularity and allow you to reuse a lot of code. Python also includes several built-in functions such as print(), but you may also construct your own.

```
def checkParity(num):

    if(num % 2 == 0):
        print("Number is even")

    else:
        print("Number is odd")

num = 5
checkParity(num)
```

Output
Number is odd

Here's a function that returns something:

```
def checkParity(num):

    if(num % 2 == 0):
        return "Number is even"
    else:
        return "Number is odd"

num = 4
parity = checkParity(num)
```

```
print(parity)
```

Output
Number is even

Exception Handling

In programming languages, exceptions are circumstances in which an error occurs that prevents the code from continuing. If you divide anything by zero, for example, a runtime exception will occur, and the program will crash. However, you may write what to do in the program if such a case arises, which is known as *exception handling*. In Python, the main code is written inside the **try** block. The exceptions are handled inside the **except** block. The **finally** block is always executed regardless of an exception occurring.

```
def divideBy(num):

    try:
        print(10 / num)

    except:
        print("Cannot divide by 0")

    finally:
        print("Division finished")

num = 0
divideBy(num)
```

Output
Cannot divide by 0
Division finished

Lists in Python

A list is a sequence of heterogeneous elements in Python. It's similar to an array, except it may hold data from several sources. The values of a changeable list can be changed. We can use indexing to parse each value of the list or to access a list element.

```
>>> list = ['truck', 'car', 'submarine', 'jet']

>>> list
['truck', 'car', 'submarine', 'jet']

>>> list = ['truck', 'car', 'submarine', 'jet']
>>> list[0]
'truck'

>>> list[1]
'car'

>>> list[2]
'submarine'

>>> list[3]
'jet'
```

We can also use negative indexes with lists:

```
>>> list = ['truck', 'car', 'submarine', 'jet']
>>> list[-2]
'submarine'
>>> list[-3]
'car'
>>> 'The {} is larger than a {}.'.format(list[-2], list[-3])
'The submarine is larger than a car.'
```

Modifying a Value of an Element in a List

```
>>> list = ['truck', 'car', 'submarine', 'jet']
>>> list[1] = 'bike'

>>> list
['cat', 'bike', 'rat', 'elephant']

>>> list[2] = list[1]
```

```
>>> list
['cat', 'bike', 'bike', 'elephant']

>>> list[-1] = 54321

>>> list
['cat', 'bike', 'bike', 54321]
```

List Concatenation and List Replication

```
>>> [4, 5, 6] + ['P', 'Q', 'R']
[4, 5, 6, 'P', 'Q', 'R']

>>> ['A', 'B', 'C'] * 4
['A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'C']

>>> list = [1, 2, 3]

>>> list = list + ['X', 'Y', 'Z']

>>> list
[1, 2, 3, 'X', 'Y', 'Z']
```

Removing Values from Lists

```
>>> list = ['truck', 'car', 'submarine', 'jet']
>>> del list[2]
>>> list
['truck', 'car', 'jet']
>>> del list[2]
>>> list
['truck', 'car']
```

Using for Loops with Lists for Traversal

```
>>> products = ['bag', 'rubber', 'knife', 'cooker']
>>> for i, product in enumerate(products):
>>>     print('Index {} in products is: {}'.format(str(i), product))

Index 0 in products is: bag
Index 1 in products is: rubber
Index 2 in products is: knife
Index 3 in products is: cooker
```

Iterating through Multiple Lists with Zip()

```
>>> name = ['David', 'Mike', 'Tommy']
>>> age = [10, 31, 54]

>>> for n, a in zip(name, age):

>>>     print('{} is {} years old'.format(n, a))

David is 10 years old

Mike is 31 years old

Tommy is 54 years old
```

The In and Not in Operators

```
>>> 'pen' in ['cap', 'owl', 'pen', 'rubber']
True
>>> list = ['cap', 'owl', 'pen', 'rubber']
>>> 'truck' in list
False
>>> 'pen' not in list
False
>>> 'train' not in list
True
```

Finding a Value in a List with the Index() Method

```
>>> list = ['notebook', 'pen', 'eraser', 'sharpener']

>>> list.index('pen')

1
```

Adding Values to Lists with the Append() and Insert() Methods

append()

```
>>> list = ['car', 'truck', 'bike']

>>> list.append('bicycle')

>>> list

['car', 'truck', 'bike', 'bicycle']
```

insert()

```
>>> list = ['car', 'truck', 'bike']

>>> list.insert(1, 'bicycle')

>>> list

['car', 'bicycle', 'truck', 'bike']
```

Removing Values from Lists with Remove()

```
>>> list = ['car', 'bike', 'submarine', 'jet']
```

```
>>> list.remove('bike')

>>> list

['car', 'submarine', 'jet']
```

If a value appears multiple times in the list, only the first instance of the value will be removed.

Sorting the Values in a List with the Sort() Method

```
>>> list = [2, 3, 1]
>>> list.sort()
>>> list
[1, 2, 3]
```

Dictionaries and Structuring Data

A Python dictionary is a collection of elements that are not in any particular order. A dictionary has a key: value pair, whereas other compound data types simply have value as an element.

The Keys(), Values(), and Items() Methods

- Traversing the values:

```
>>> book = {'color': 'red', 'price': 160}
>>> for v in book.values():
>>>     print(v)
red
160
```

- Traversing the keys:

```
>>> for k in book.keys():
```

```
>>>     print(k)
color
price
```

- Traversing keys and values:

```
>>> for i in book.items():
>>>     print(i)
('color', 'red')
('price', 160)
```

A for loop can iterate through the keys, values, or key-value pairs in a dictionary using the **keys**(), **values**(), and **items**() methods, respectively.

The Get() Method

Get() accepts two parameters: a key and a default value if the key isn't found.

```
>>> items = {'chairs': 5, 'tables': 2}

>>> 'There are {} tables.'.format(str(items.get('tables', 0)))
'There are 2 tables.'

>>> 'There are {} computers.'.format(str(items.get('computers', 0)))
'There are 0 computers.'
```

Check Key's Presence in Dictionary

```
>>> 'color' in book
True
```

Sets

A set is an unordered collection of unique elements. Python sets are similar to mathematics sets, and allow all set related operations including union, intersection, and difference.

Creating a Set

You can generate sets by using curly braces {} and the built-in function **set** ():

```
>>> s = {2, 4, 6}
>>> s = set([2, 4, 6])
```

If you use curly braces {} to create an empty set, you'll get the data structure as a **dictionary** instead.

```
>>> s = {}
>>> type(s)
<class 'dict'>
```

All duplicate values are automatically removed by a set:

```
>>> s = {1, 2, 3, 2, 3, 4, 4, 5}
>>> s
{1, 2, 3, 4, 5}
```

Adding to the Set

```
>>> a = {1, 2, 3, 4, 5}
>>> a.add(6)
>>> a
{1, 2, 3, 4, 5, 6}

>>> set = {0, 1, 2, 3, 4}
>>> set.update([2, 3, 4, 5, 6])
>>> set
{0, 1, 2, 3, 4, 5, 6}
```

Removing from a Set

The **remove**() and **discard**() methods remove an element from the set; however **remove**() will throw a key error if the value isn't present.

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

```
>>> set.remove(4)
>>> set
{1, 2, 3}
>>> set.remove(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 3
```

You can also use **discard**():

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

Union of Multiple Sets

```
>>> s1 = {1, 2, 3, 4}
>>> s2 = {3, 4, 5, 6}
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6}
```

Intersection of Multiple Sets

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

Difference of Two Sets

```
>>> s1 = {1, 2, 3}
>>> s2 = {2, 3, 4}
```

```
>>> s1.difference(s2)
{1}
>>> s2.difference(s1)
{4}
```

Symmetric Difference of Two Sets

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

itertools Module

When dealing with iterators, the itertools module offers a set of quick and memory-efficient tools (like lists or dictionaries).

Accumulate()

Using accumulate() returns the results of a function as an iterator:

```
import itertools
import operator

data = [1, 2, 3, 4, 5]
result = itertools.accumulate(data, operator.mul)
for each in result:
    print(each)
```

Output
1
2
6
24
120

The **operator.mul()** takes two numbers and multiplies them:

```
operator.mul(3, 5)
15
operator.mul(4, 3)
12
operator.mul(6, 3)
18
operator.mul(2, 5)
10
```

We can also use the method *without* any iterator:

```
import itertools

data = [1, 2, 3, 4, 5, 6, 7]
result = itertools.accumulate(data)
for each in result:
    print(each)
```

Output
1
3
6
10
15
21
28

Combinations()

```
import itertools

shapes = [1, 2, 3, 4, 5]
combinations = itertools.combinations(shapes, 2)
```

```
for combination in combinations:
    print(combination)
```

Output
(1, 2)
(1, 3)
(1, 4)
(1, 5)
(2, 3)
(2, 4)
(2, 5)
(3, 4)
(3, 5)
(4, 5)

Combinations_with_Replacement()

```
import itertools

shapes = [1, 2, 3, 4, 5]
combinations = itertools.combinations_with_replacement(shapes, 2)
for combination in combinations:
    print(combination)
```

Output
(1, 1)
(1, 2)
(1, 3)
(1, 4)
(1, 5)
(2, 2)
(2, 3)
(2, 4)
(2, 5)

(3, 3)
(3, 4)
(3, 5)
(4, 4)
(4, 5)
(5, 5)

Count()

A count takes the initial point and step size:

```
import itertools

for i in itertools.count(1, 3):
    print(i)
    if i >= 15:
        break
```

Output
1
4
7
10
13
16

Cycle()

Here is an itertools.cycle(iterable):

```
import itertools

arr = [1, 2, 3, 4, 5]
c = 0
for itr in itertools.cycle(arr):
```

```
    if(c > 20):
        break
    print(itr)
    c += 1
```

Output
1
2
3
4
5
1
2
3
4
5
1
2
3
4
5
1
2
3
4
5
1

Comprehensions

Dictionary Comprehension

```
>>> dict = {1: 2, 3: 4, 5: 6}
>>> {value: key for key, value in dict.items()}
```

```
{2: 1, 4: 3, 6: 5}
```

List Comprehension

```
>>> a= [i for i in range(1, 20)]
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Set Comprehension

```
>>> a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
>>> a = [i+1 for i in a]
>>> a
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Lambda Functions

Lambda functions are one-liner functions of Python:

```
>>> add = lambda a, b: a + b
>>> add(1, 2)
3
```

String Formatting

Using % Operator

```
>>> a = 4
>>> 'Value is %x' % a
'Value is 4'
```

.format()

```
>>> a = 4
>>> 'Value is {}'.format(a)
'Value is 4'
```

Formatted String Literals (F-Strings)

```
>>> a = 4
>>> f'Value is {a}'
'Value is 4'
```

Ternary Conditional Operator

We can write the conditional operators **if** and **else** in the single line using the ternary conditional operator:

```
>>> a = 5
>>> print('Number is even' if a % 2 ==0 else 'Number is odd')
Number is odd
>>>
```