Real Python: Python 3 Cheat Sheet

Contents

Chapter 1

Introduction

Python is a beautiful language. It's easy to learn and fun, and its syntax is simple yet elegant. Python is a popular choice for beginners, yet still powerful enough to back some of the world's most popular products and applications from companies like NASA, Google, Mozilla, Cisco, Microsoft, and Instagram, among others. Whatever the goal, Python's design makes the programming experience feel almost as natural as writing in English.

Check out **Real Python** to learn more about Python and web development. Email your questions or feedback to **info@realpython.com**.

Chapter 2

Primitives

Numbers

Python has integers and floats. Integers are simply whole numbers, like 314, 500, and 716. Floats, meanwhile, are fractional numbers like 3.14, 2.867, 76.88887. You can use the type method to check the value of an object.

```
>>> type(3)
<class 'int'>
>>> type(3.14)
<class 'float'>
>>> pi = 3.14
>>> type(pi)
<class 'float'>
```

In the last example, `pi` is the variable name, while `3.14` is the value.

You can use the basic mathematical operators:

```
>>> 3 + 3
6
>>> 3 - 3
0
>>> 3 / 3
1.0
>>> 3 / 2
```

```
1.5
>>> 3 * 3
9
>>> 3 ** 3
27
>>> num = 3
>>> num = num - 1
>>> print(num)
2
>>> num = num + 10
>>> print(num)
12
>>> num += 10
>>> print(num)
22
>>> num -= 12
>>> print(num)
10
>>> num *= 10
>>> num
100
```

There's also a special operator called modulus, `%` , that returns the remainder after integer division.

```
>>> 10 % 3
1
```

One common use of modulus is determining if a number is divisible by another number. For example, we know that a number is even if it's divided by 2 and the remainder is 0.

```
>>> 10 % 2
0
>>> 12 % 2
0
```

Finally, make sure to use parentheses to enforce precedence.

```
>>> (2 + 3) * 5
25
>>> 2 + 3 * 5
17
```

Strings

Strings are used quite often in Python. Strings, are just that, a string of characters - which s anything you can type on the keyboard in one keystroke, like a letter, a number, or a backslash.

Python recognizes single and double quotes as the same thing, the beginning and end of the strings.

```
>>> "string list"
'string list'
>>> 'string list'
'string list'
```

What if you have a quote in the middle of the string? Python needs help to recognize quotes as part of the English language and not as part of the Python language.

```
>>> "I 'cant do that"
'I 'cant do that'
>>> "He said \"no\" to me"
'He said "no" to me'
```

Now you can also join (concatenate) strings with use of variables as well.

```
>>> a = "first"
>>> b = "last"
>>> a + b
'firstlast'
```

If you want a space in between, you can change a to the word with a space after.

```
>>> a = "first "
>>> a + b
'first last'
```

There are different string methods for you to choose from as well - like `upper()`, `lower()`, `replace()`, and `count()`.

`upper()` does just what it sounds like - changes your string to all uppercase letters.

```
>>> str = 'woah!'
>>> str.upper()
'WOAH!'
```

Can you guess what `lower()` does?

```
>>> str = 'WOAH!'
>>> str.lower()
'woah!'
```

`replace()` allows you to replace any character with another character.

```
>>> str = 'rule'
>>> str.replace('r', 'm')
'mule'
```

Finally, `count()` lets you know how many times a certain character appears in the string.

```
>>> number_list =['one', 'two', 'one', 'two', 'two']
>>> number_list.count('two')
3
```

You can also format/create strings with the `format()` method.

```
>>> "{0} is a lot of {1}".format("Python", "fun!")
'Python is a lot of fun!'
```

Booleans

Boolean values are simply True or False .

Check to see if a value is equal to another value with two equal signs.

```
>>> 10 == 10
True
>>> 10 == 11
False
>>> "jack" == "jack"
True
>>> "jack" == "jake"
False
```

To check for inequality use `!=`.

```
>>> 10 != 10
False
>>> 10 != 11
True
>>> "jack" != "jack"
False
>>> "jack" != "jake"
True
```

You can also test for > , < , >= , and <=.

```
>>> 10 > 10
False
>>> 10 < 11
True
>>> 10 >= 10
True
>>> 10 <= 11
True
>>> 10 <= 10 < 0
False
>>> 10 <= 10 < 11
True
>>> "jack" > "jack"
False
```

```
>>> "jack" >= "jack"
True
```

Chapter 3

Collections

Lists

Lists are containers for holding values.

```
>>> fruits = ['apple','lemon','orange','grape']
>>> fruits
['apple', 'lemon', 'orange', 'grape']
```

To access the elements in the list you can use their associated index value. Just remember that the list starts with `0`, not `1`.

```
>>> fruits[2]‘’
orange
```

If the list is long and you need to count from the end you can do that as well.

```
>>> fruits[-2]‘’
orange
```

Now, sometimes lists can get long and you want to keep track of how many elements you have in your list. To find this, use the `len()` function.

```
>>> len(fruits)
4
```

Use `append()` to add a new element to the end of the list and `pop()` to remove an element from the end.

```
>>> fruits.append('blueberry')
>>> fruits
['apple', 'lemon', 'orange', 'grape', 'blueberry']
>>> fruits.append('tomato')
>>> fruits
['apple', 'lemon', 'orange', 'grape', 'blueberry', 'tomato']
>>> fruits.pop()
'tomato'
>>> fruits
['apple', 'lemon', 'orange', 'grape', 'blueberry']
```

Check to see if a value exists using in the list.

```
>>> 'apple' in fruits
True
>>> 'tomato' in fruits
False
```

Dictionaries

A dictionary optimizes element lookups. It uses key/value pairs, instead of numbers as placeholders. Each key must have a value, and you can use a key to look up a value.

```
>>> words = {'apple': 'red','lemon': 'yellow'}
>>> words
{'apple': 'red', 'lemon': 'yellow'}
>>> words['apple']
'red'
>>> words['lemon']
'yellow'
```

This will also work with numbers.

```
>>> dict = {'one': 1, 'two': 2}
>>> dict
{'one': 1, 'two': 2}
```

Output all the keys with `keys()` and all the values with `values()`.

```
>>> words.keys()
dict_keys(['apple', 'lemon'])
>>> words.values()
dict_values(['red', 'yellow'])
```

Chapter 4

Control Statements

IF Statements

The IF statement is used to check if a condition is true.

Essentially, if the condition is true, the Python interpreter runs a block of statements called the if-block. If the statement is false, the interpreter skips the if block and processes another block of statements called the else-block. The else clause is optional.

Let's look at two quick examples.

```
>>> num = 20
>>> if num == 20:
...     print('the number is 20')
... else:
...     print('the number is not 20')
...
the number is 20
>>> num = 21
>>> if num == 20:
...     print('the number is 20')
... else:
...     print('the number is not 20')
...
the number is not 20
```

You can also add an elif clause to add another condition to check for.

```
>>> num = 21
```

```
>>> if num == 20:
...     print('the number is 20')
... elif num > 20:
...     print('the number is greater than 20')
... else:
... print('the number is less than 20')
...
the number is greater than 20
```

Loops

There are 2 kinds of loops used in Python - the `for` loop and the `while` loop. `for` loops are traditionally used when you have a piece of code which you want to repeat n number of times. They are also commonly used to loop or iterate over lists.

```
>>> colors = ['red', 'green', 'blue']
>>> colors
['red', 'green', 'blue']
>>> for color in colors:
...     print('I love ' + color)
...
I love red
I love green
I love blue
```

`while` loops, like the `for` Loop, are used for repeating sections of code - but unlike a `for` loop, the `while` loop continues until a defined condition is met.

```
>>> num = 1
>>> num
1
>>> while num <= 5:
...     print(num)
...     num += 1
...
1
2
3
4
5
```

Chapter 5

Functions

Functions are blocks of reusable code that perform a single task.

You use `def` to define (or create) a new function then you call a function by adding parameters to the function name.

```
>> def multiply(num1, num2):
...     return num1 * num2
...
>>> multiply(2, 2)
4
```

You can also set default values for parameters.

```
>>> def multiply(num1, num2=10):
...     return num1 * num2
...
>>> multiply(2)
20
```

Ready to learn more? Visit **Real Python** to learn Python and web development. Cheers!