

The search() Function

The search() function searches the string for a match, and returns a [Match object](#) if there is a match.

If there is more than one match, only the first occurrence of the match will be returned:

Example

Search for the first white-space character in the string:

```
import re
```

```
txt = "The rain in Spain"
```

```
x = re.search("\s", txt)
```

```
print("The first white-space character is located in position:", x.start())
```

If no matches are found, the value None is returned:

Example

Make a search that returns no match:

```
import re
```

```
txt = "The rain in Spain"
```

```
x = re.search("Portugal", txt)
```

```
print(x)
```

The split() Function

The split() function returns a list where the string has been split at each match:

Example

Split at each white-space character:

```
import re
```

```
txt = "The rain in Spain"
```

```
x = re.split("\s", txt)
```

```
print(x)
```

You can control the number of occurrences by specifying the maxsplit parameter:

Example

Split the string only at the first occurrence:

```
import re

txt = "The rain in Spain"
x = re.split("\s", txt, 1)
print(x)
```

The sub() Function

The sub() function replaces the matches with the text of your choice:

Example

Replace every white-space character with the number 9:

```
import re

txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

You can control the number of replacements by specifying the count parameter:

Example

Replace the first 2 occurrences:

```
import re

txt = "The rain in Spain"
x = re.sub("\s", "9", txt, 2)
print(x)
```

Python PIP

What is PIP?

PIP is a package manager for Python packages, or modules if you like.

Note: If you have Python version 3.4 or later, PIP is included by default.

What is a Package?

A package contains all the files you need for a module.

Modules are Python code libraries you can include in your project.

Check if PIP is Installed

Navigate your command line to the location of Python's script directory, and type the following:

Example

Check PIP version:

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip --version
```

Install PIP

If you do not have PIP installed, you can download and install it from this page: <https://pypi.org/project/pip/>

Download a Package

Downloading a package is very easy.

Open the command line interface and tell PIP to download the package you want.

Navigate your command line to the location of Python's script directory, and type the following:

Example

Download a package named "camelcase":

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip install camelcase
```

Python Try Except

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The else block lets you execute code when there is no error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

Exception Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the try statement:

Example

The try block will generate an exception, because x is not defined:

```
try:
    print(x)
except:
    print("An exception occurred")
```

Since the try block raises an error, the except block will be executed.

Without the try block, the program will crash and raise an error:

Example

This statement will raise an error, because x is not defined:

```
print(x)
```

Many Exceptions

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

Example

Print one message if the try block raises a `NameError` and another for other errors:

```
try:
    print(x)
except NameError:
    print("Variable x is not defined")
except:
    print("Something else went wrong")
```

Else

You can use the `else` keyword to define a block of code to be executed if no errors were raised:

Example

In this example, the try block does not generate any error:

```
try:
    print("Hello")
except:
    print("Something went wrong")
else:
    print("Nothing went wrong")
```

Finally

The `finally` block, if specified, will be executed regardless if the try block raises an error or not.

Example

```
try:
    print(x)
except:
    print("Something went wrong")
```

finally:

```
print("The 'try except' is finished")
```

This can be useful to close objects and clean up resources:

Example

Try to open and write to a file that is not writable:

try:

```
f = open("demofile.txt")
```

try:

```
f.write("Lorum Ipsum")
```

except:

```
print("Something went wrong when writing to the file")
```

finally:

```
f.close()
```

except:

```
print("Something went wrong when opening the file")
```

The program can continue, without leaving the file object open.

Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword.

Example

Raise an error and stop the program if x is lower than 0:

```
x = -1
```

if x < 0:

```
raise Exception("Sorry, no numbers below zero")
```

The raise keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

Example

Raise a `TypeError` if `x` is not an integer:

```
x = "hello"
```

if not `type(x)` is `int`:

```
    raise TypeError("Only integers are allowed")
```