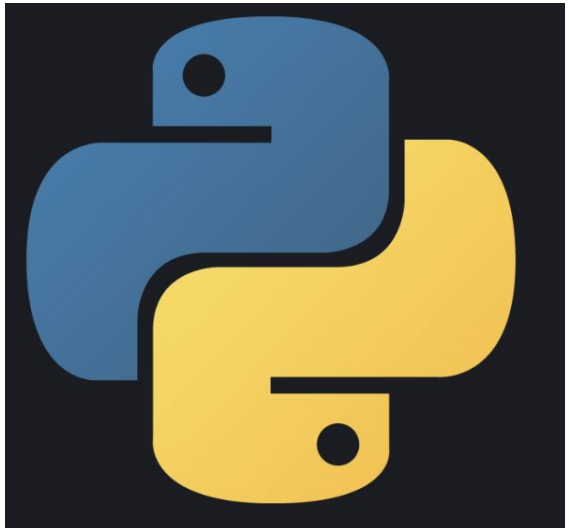


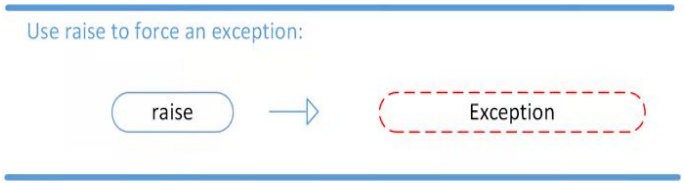
Exceptions in Python

Raising an Exception in Python - Intro

There are scenarios where you might want to stop your program by **raising** an exception if a **condition** occurs.

You can do this with the **raise** keyword

You can even complement the statement with a custom message.

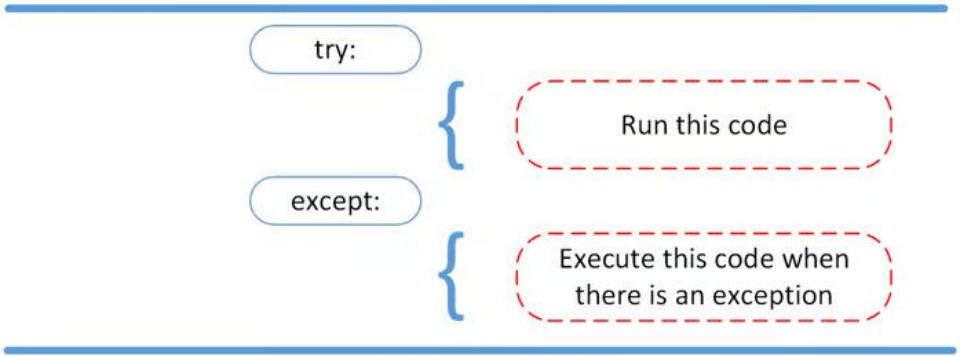



```
number = 10
if number > 5:
    raise Exception(f"The number should not exceed 5. ({number=})")
print(number)
```

Handling Exceptions With the `try` and `except` Block

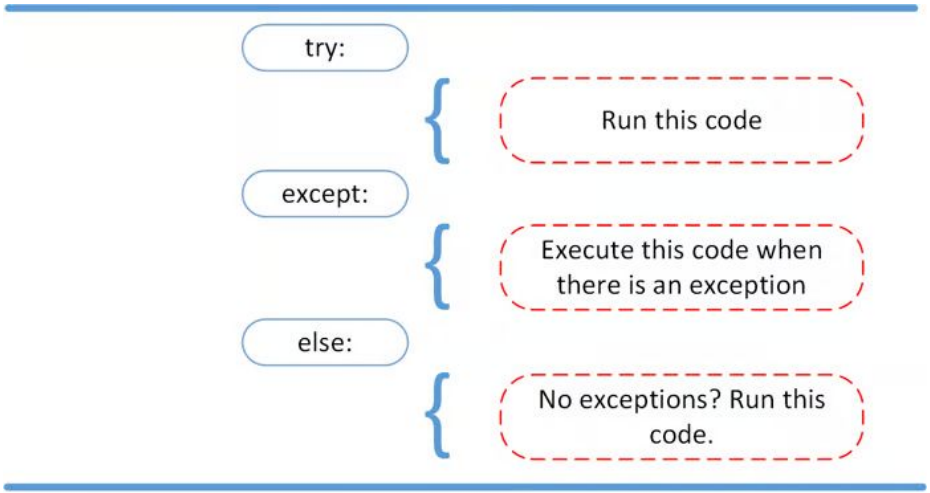
In Python, you use the **try** and **except** block to catch and handle exceptions.

Python executes code following the try statement as a normal part of the program. The code that follows the except statement is the program's response to any exceptions in the preceding try clause.



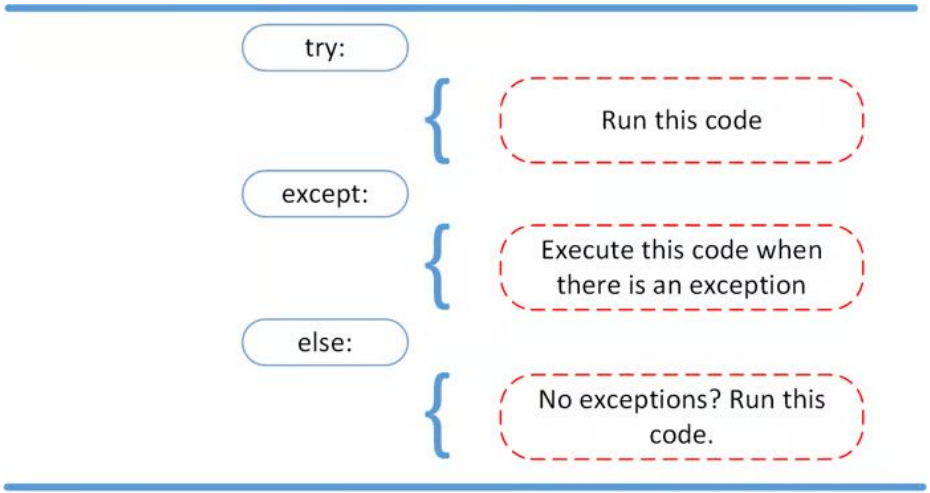
Proceeding After a Successful Try With else

You can use Python's else statement to instruct a program to execute a certain block of code only in the absence of exceptions:



Proceeding After a Successful Try With else

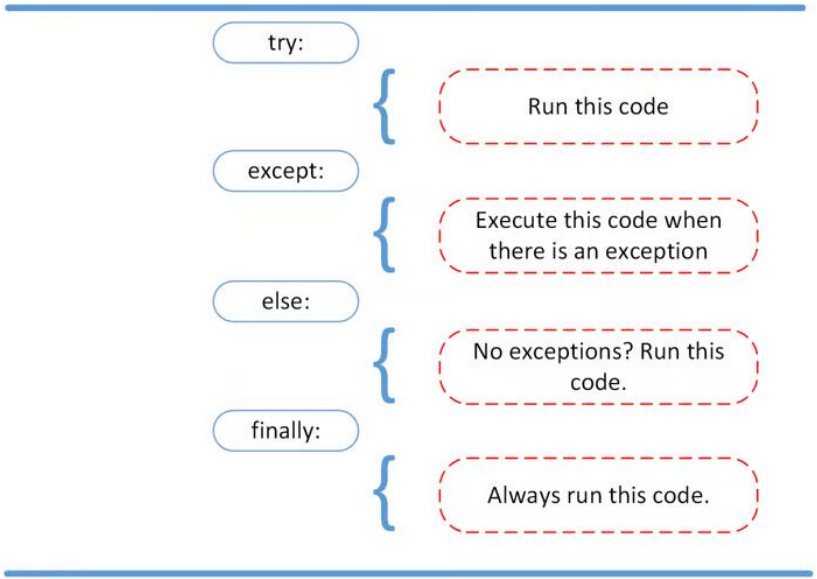
You can use Python's else statement to instruct a program to execute a certain block of code only in the absence of exceptions:




```
try:
    linux_interaction()
except RuntimeError as error:
    print(error)
else:
    print("Doing even more Linux things.")
```

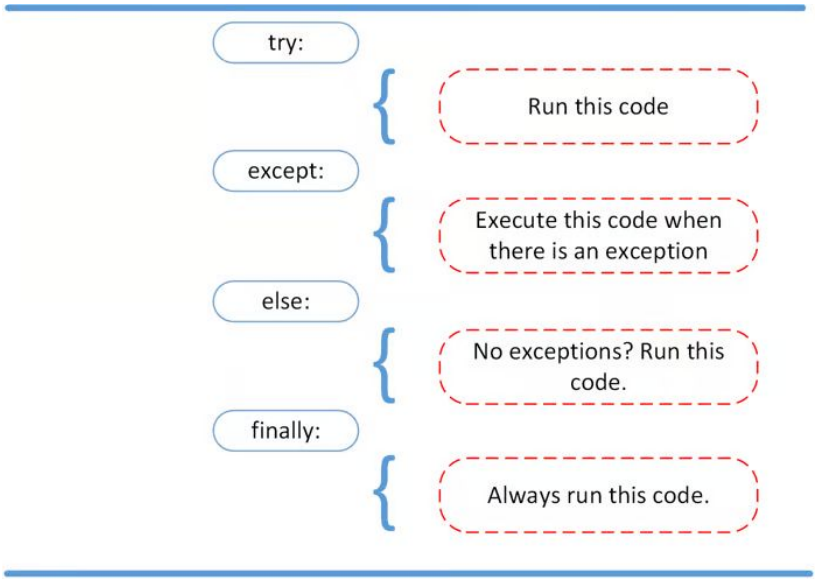
Cleaning Up After Execution With `finally`

Imagine that you always had to implement some sort of action to clean up after executing your code. Python enables you to do so using the `finally` clause



Cleaning Up After Execution With `finally`

Imagine that you always had to implement some sort of action to clean up after executing your code. Python enables you to do so using the `finally` clause




```
try:
    linux_interaction()
except RuntimeError as error:
    print(error)
else:
    try:
        with open("file.log") as file:
            read_data = file.read()
    except FileNotFoundError as fnf_error:
        print(fnf_error)
finally:
    print("Cleaning up, irrespective of any exceptions.")
```

Creating Custom Exceptions in Python

With the large number of built-in exceptions that Python offers, you'll likely find a fitting type when deciding which exception to raise.

However, sometimes your code won't fit the mold.

Python makes it straightforward to create custom exception types by inheriting from a built-in exception.

Creating Custom Exceptions in Python

You generally create a custom exception in Python by inheriting from Exception.

That is the base class for most built-in Python exceptions as well.

You could also inherit from a different exception, but choosing Exception is usually the best choice.

```
class PlatformException(Exception):
    """Incompatible platform."""
```