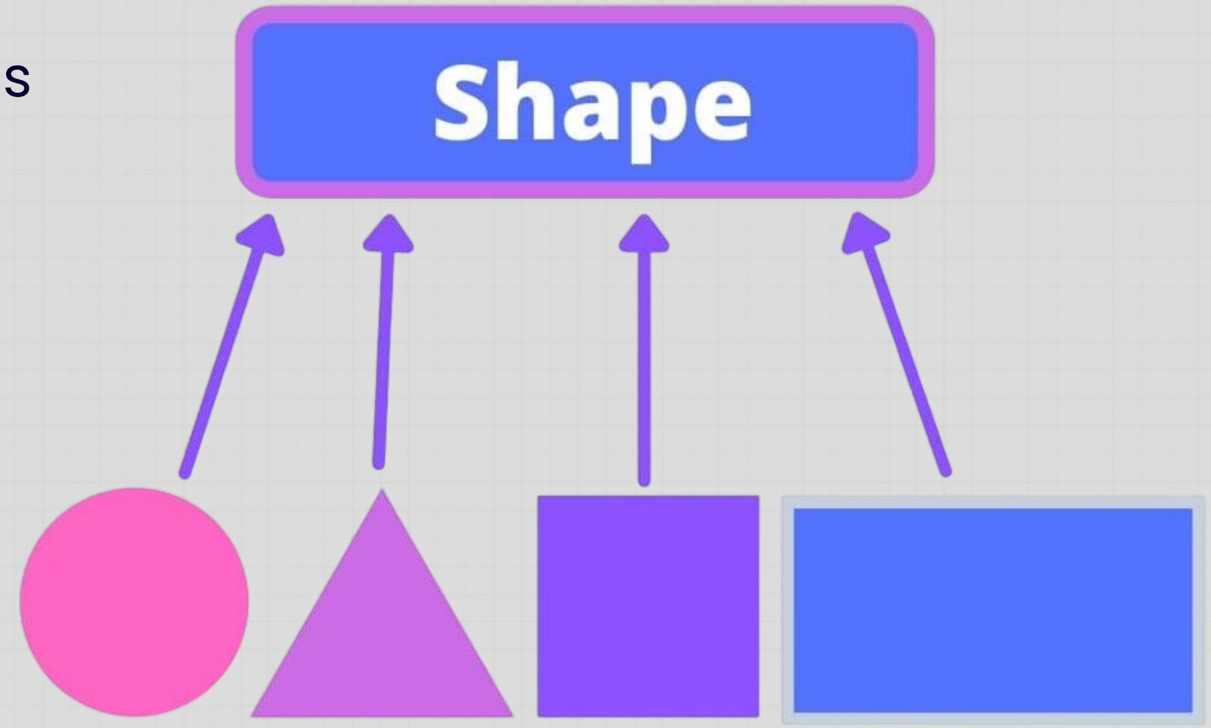
CYDEO

Day04 Presentation SLides

Contents

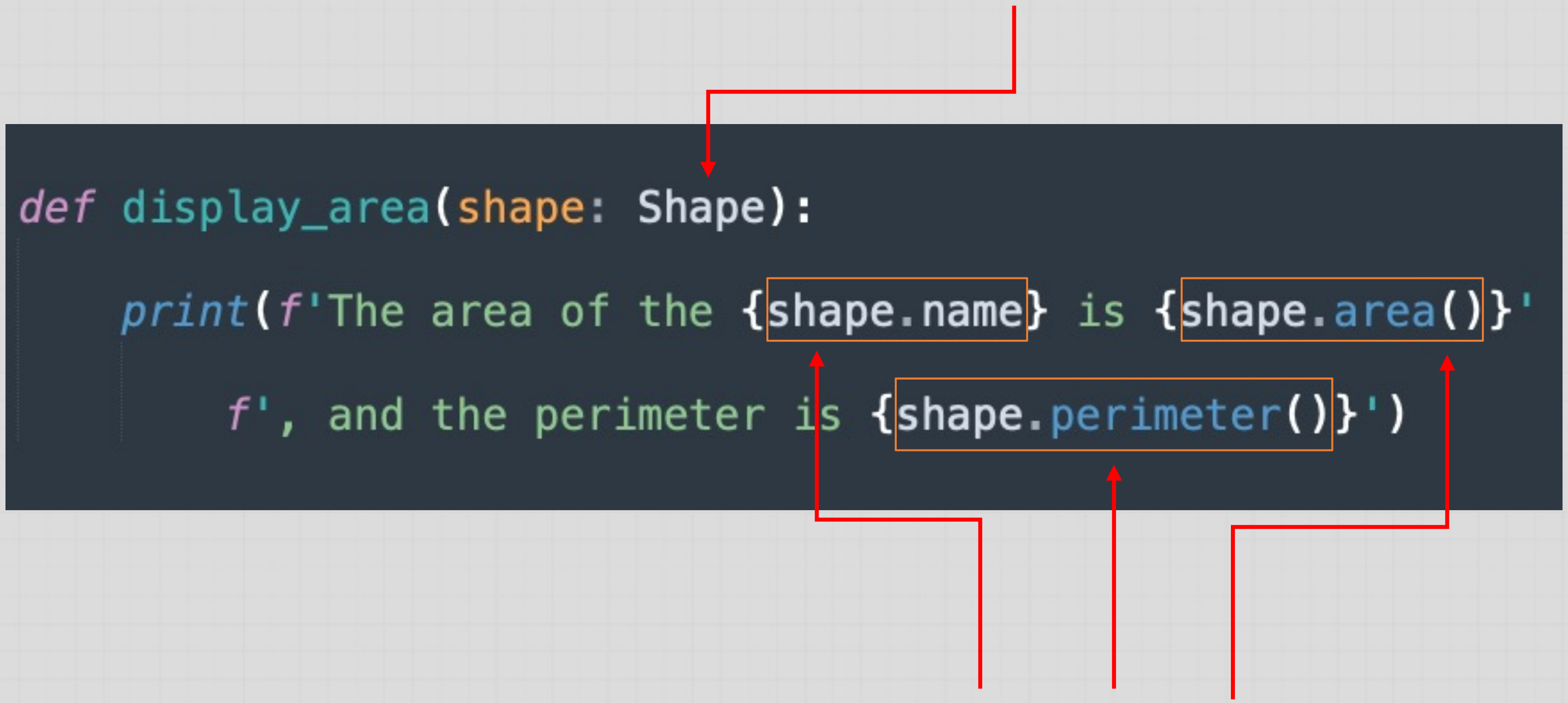
Polymorphism

- Ability of the object to take on many forms
- Allows the objects of different classes to be treated as the same type
- The inheritance relationship between those different classes are required



Polymorphism

The parameter's data type is set to Shape. it will the argument object that's passed to function to be an instance of any shape class

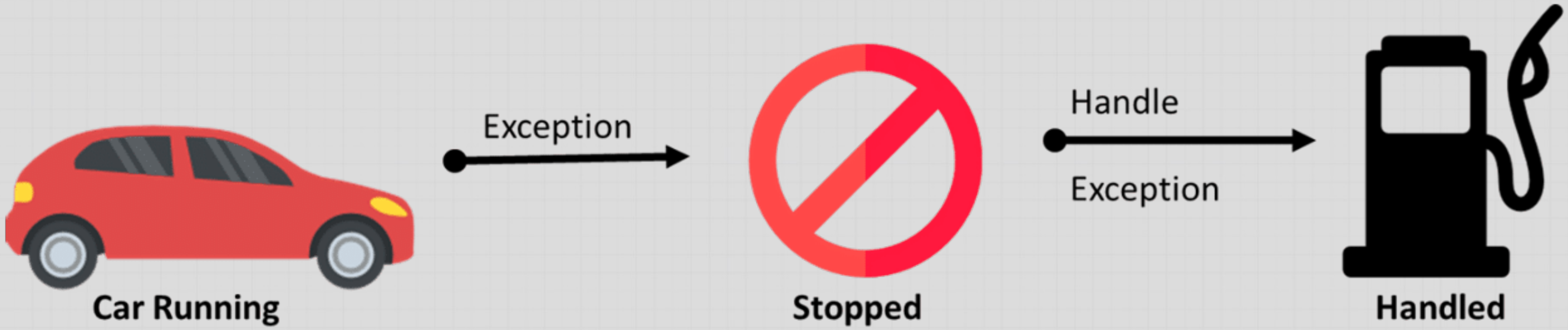


```
def display_area(shape: Shape):
    print(f'The area of the {shape.name} is {shape.area()}'
          f', and the perimeter is {shape.perimeter()}')
```

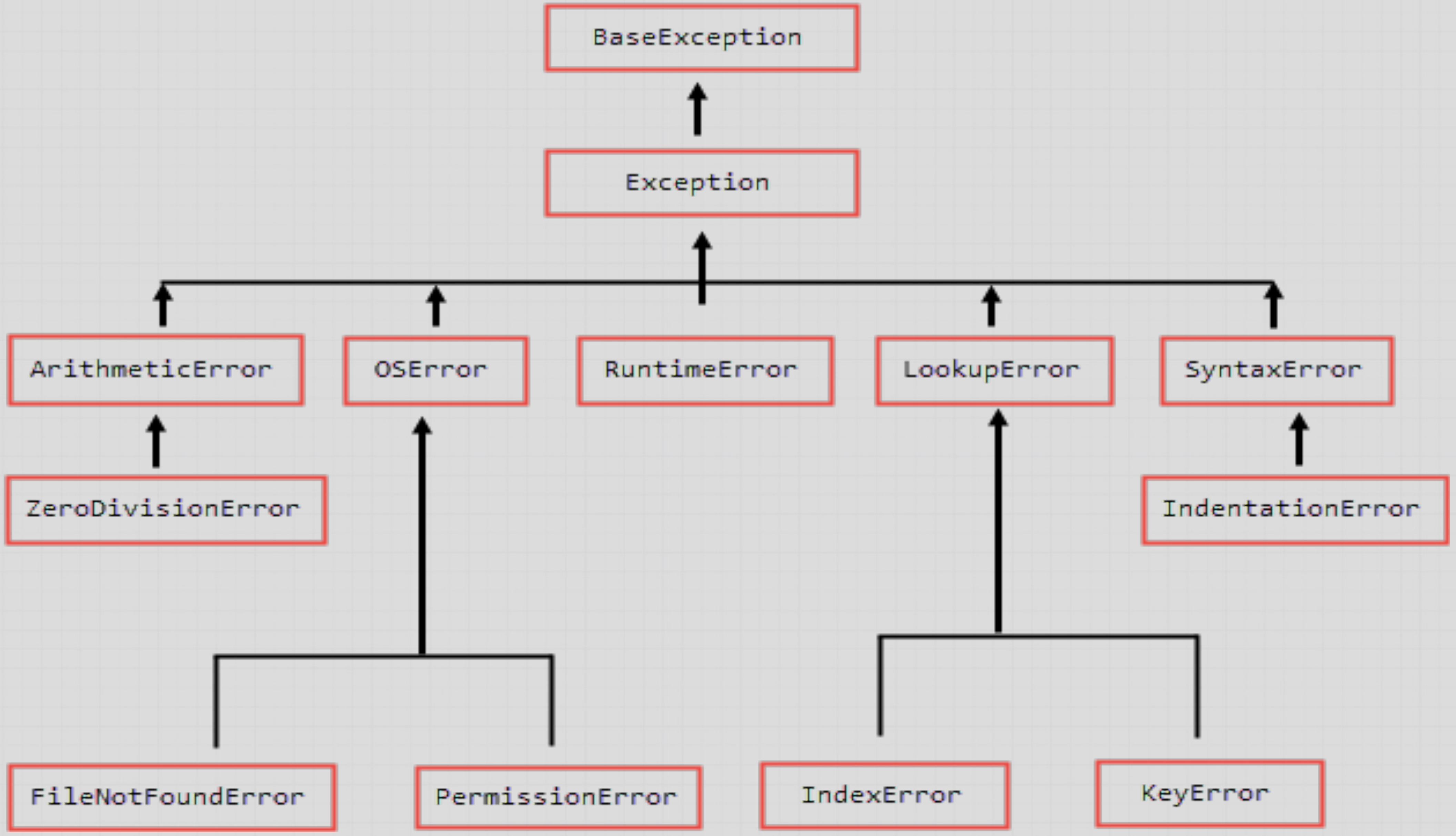
The common methods and variables of all the different shape objects

Exceptions

- An unwanted or unexpected event (Something went wrong)

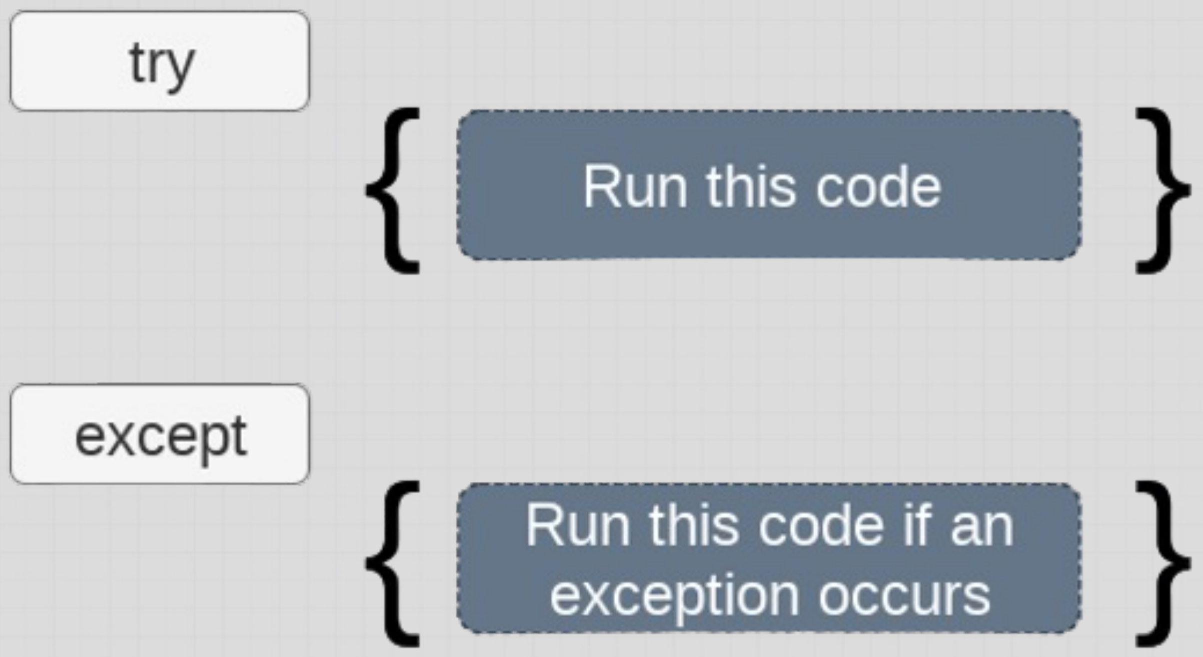


Exceptions Hierarchy



Exception Handling

- To prevent exceptions from crashing our program, we must write code that detects and handles them



Exception Handling – try & except

- To handle an exception, we can use try & except blocks

```
try:
    # try block statements
    # some code that might throw exception

except:
    # except block statements
    # handles exception
```

Multiple except blocks can be given if we specify the type of exception in each except block

Exception Handling – else

- An optional block that can be given after the except block
- Gets executed if there is no exception occurred in try block

```
try:
    # try block statements
except:
    # except block statements
else:
    # else block statemnents
```

Exception Handling – finally

- An optional block that can be given after the last block
- Always executed after try & except blocks whether an exception occurs or not

```
try:
    # try block statements
except:
    # except block statements
else:
    # else block statemnents
finally:
    # finally block statements
```

Raise an Exception

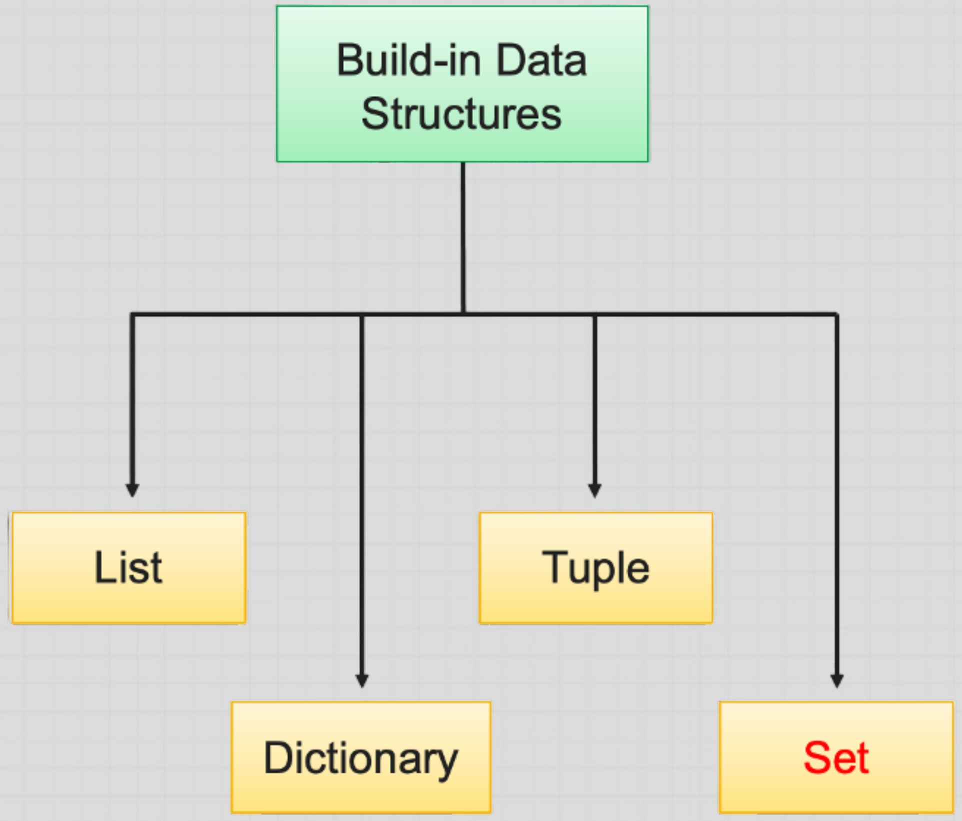
- We can choose to manually throw an exception if a condition occurs
- The raise keyword is used for throwing an exception

```python
if age < 0:
    raise Exception('Age can not be negative')
```

```python
if len(name) == 0:
    raise RuntimeError('name can not be empty')
```

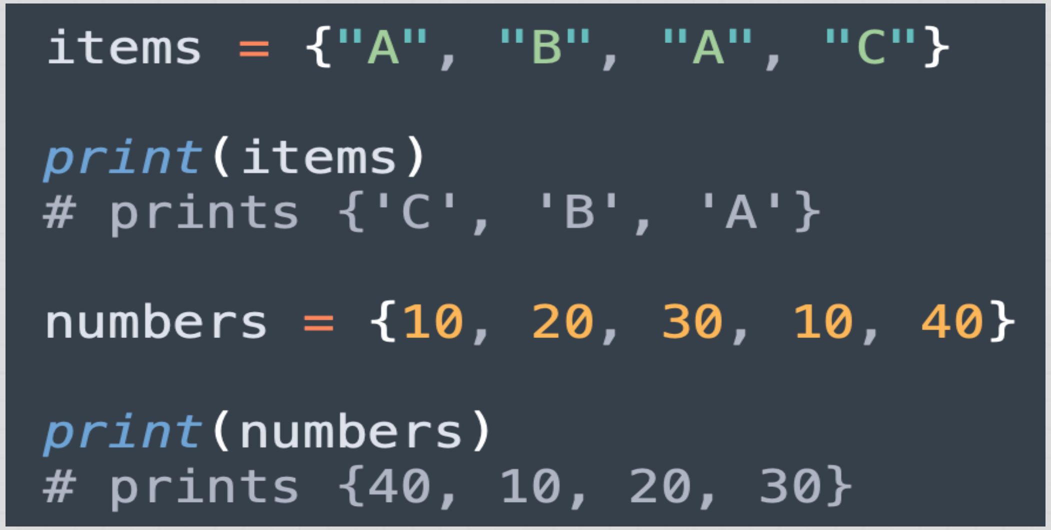
Set

- A special type of variable
- Used to store multiple unique values
- Size is dynamic, and can be increased/decreased
- The elements in the set are unchangeable
- Elements in the set do not have index numbers



Creating Set

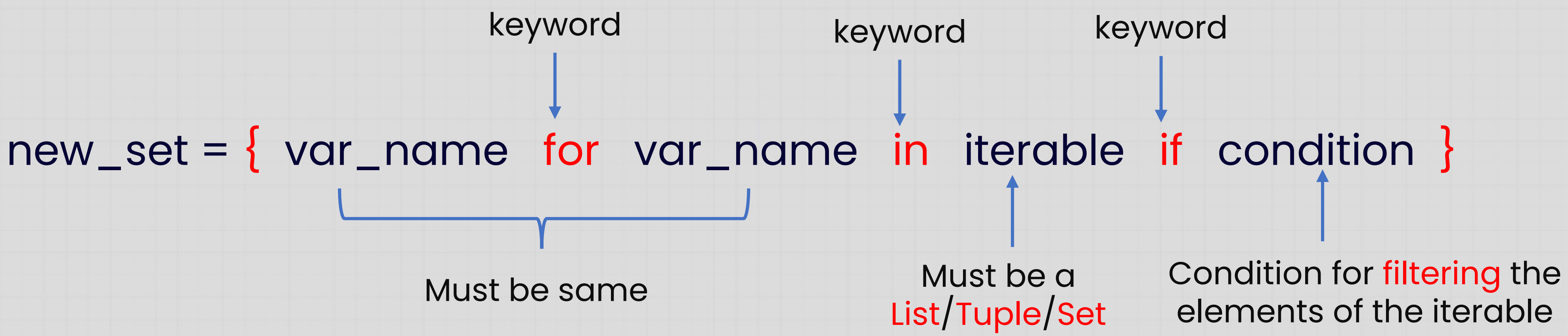
- Created by placing all the elements inside curly brackets { } separated by commas
- Elements in the set are unordered, unchangeable and can be of any data type
- The Set does not accept duplicated elements

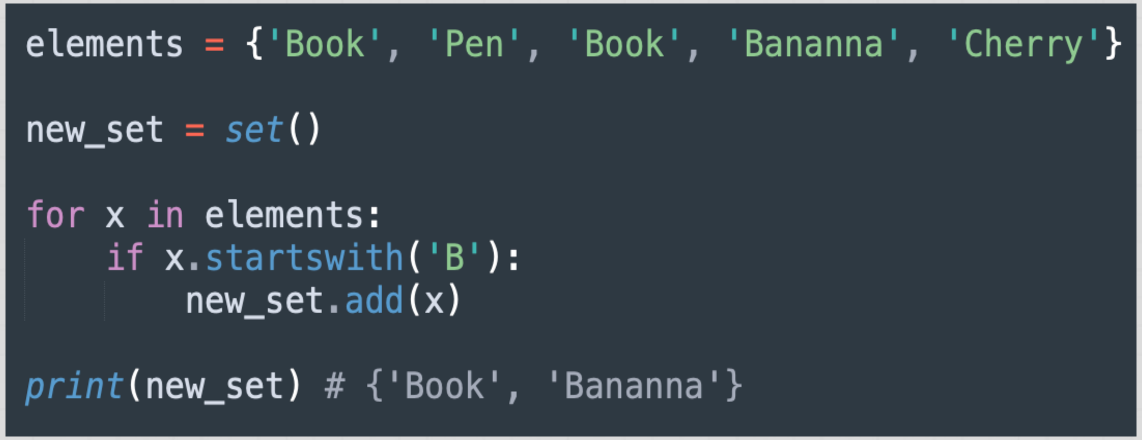


```
items = {"A", "B", "A", "C"}

print(items)
# prints {'C', 'B', 'A'}

numbers = {10, 20, 30, 10, 40}

print(numbers)
# prints {40, 10, 20, 30}
```

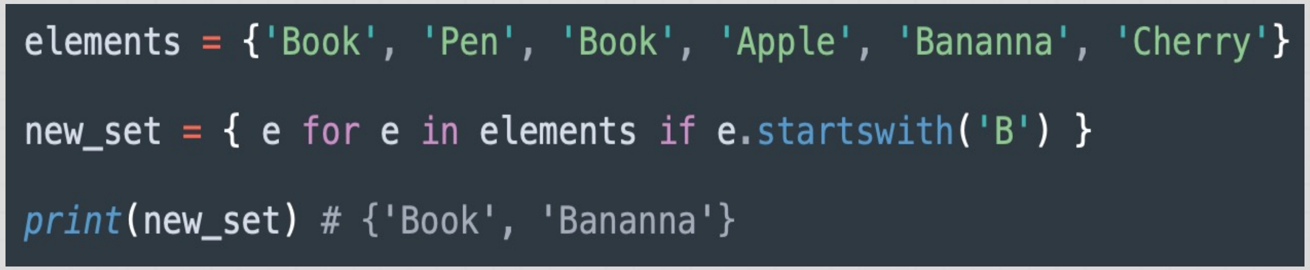
Set Comprehensions

- Used to create a new set based the values of an exiting iterable (set/list/tuple)


keyword
keyword
keyword
new_set = { var_name for var_name in iterable if condition }
Must be same
Must be a List/Tuple/Set
Condition for filtering the elements of the iterable



```python
elements = {'Book', 'Pen', 'Book', 'Bananna', 'Cherry'}

new_set = set()

for x in elements:
    if x.startswith('B'):
        new_set.add(x)

print(new_set) # {'Book', 'Bananna'}
```



```python
elements = {'Book', 'Pen', 'Book', 'Apple', 'Bananna', 'Cherry'}

new_set = { e for e in elements if e.startswith('B') }

print(new_set) # {'Book', 'Bananna'}
```

Set Methods

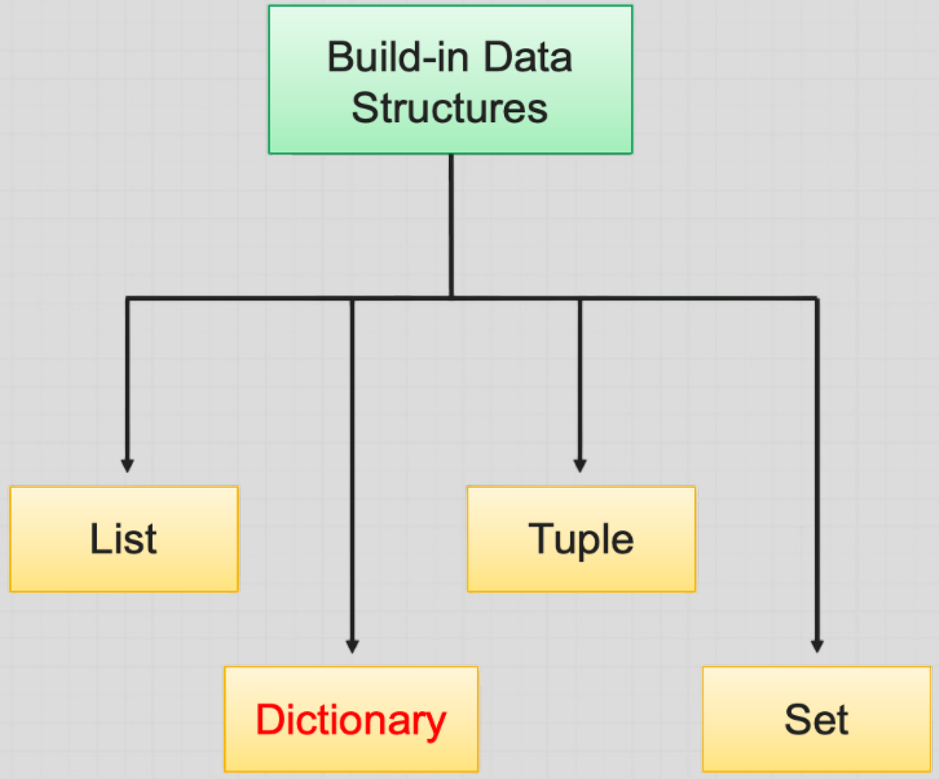
Method Name	Method Name	Method Name
add()	remove()	clear()
update()	pop()	copy()
difference()	intersection()	different_update()
intersection_update()	symmetric_update	

Tuple vs List vs Set

Tuple	List	Set
Created by using ( ) or tuple() function	Created by using [ ] or list() function	Created by using { } or set() function
Faster	Slower	Slower
Size is fixed	Size is dynamic	Size is dynamic
Indexing/Slicing is allowed	Indexing/Slicing is allowed	Indexing/Slicing is NOT allowed
Ordered	Ordered	Unordered
Elements are unchangeable	Elements are changeable	Elements are unchangeable
Duplicates are allowed	Duplicates are allowed	Duplicates are NOT allowed

Dictionary

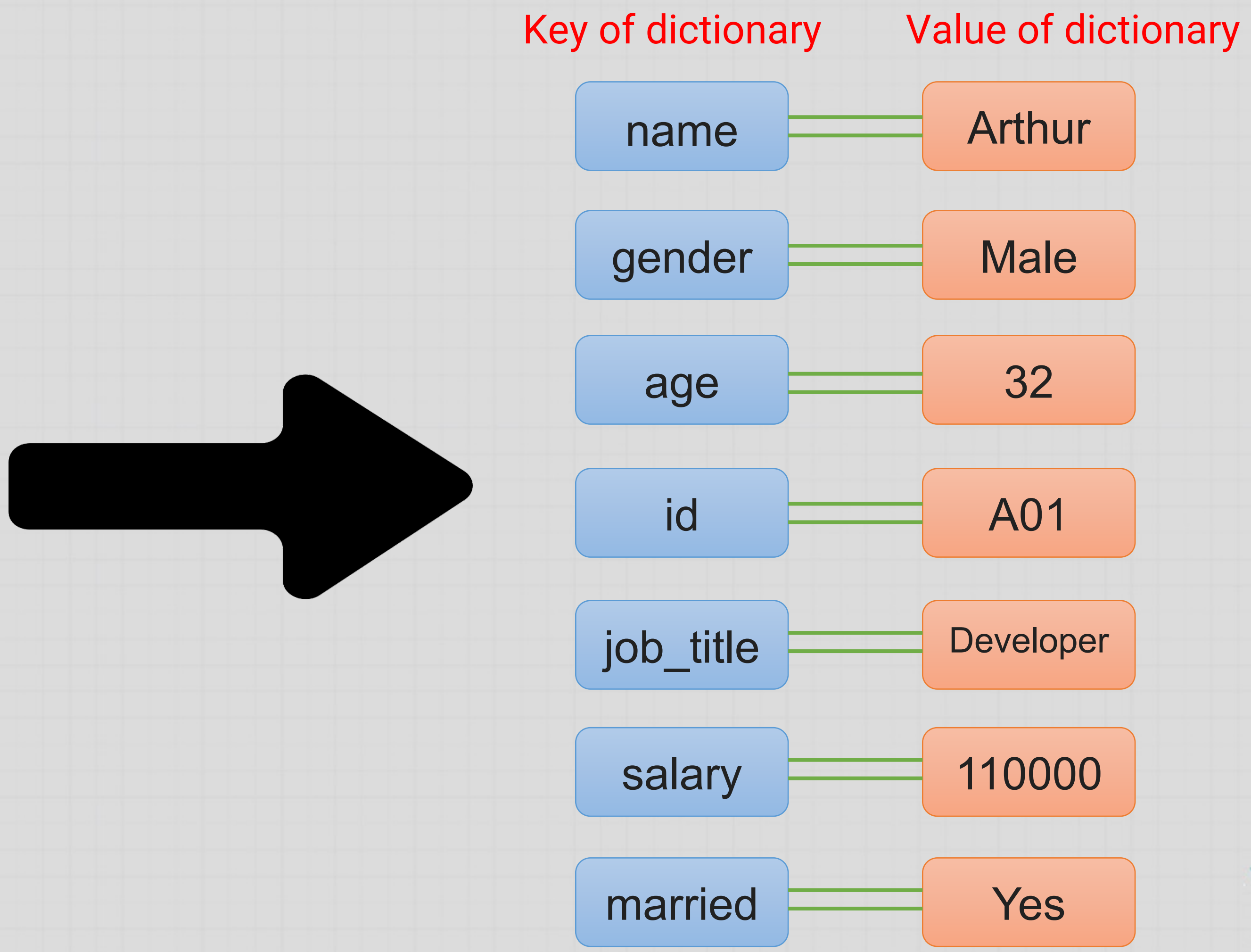
- Collection of pairs
- Data structure based on the key + value pairs
- Size is dynamic, and can be increased/decreased
- The Items in the dictionary are changeable



Dictionary

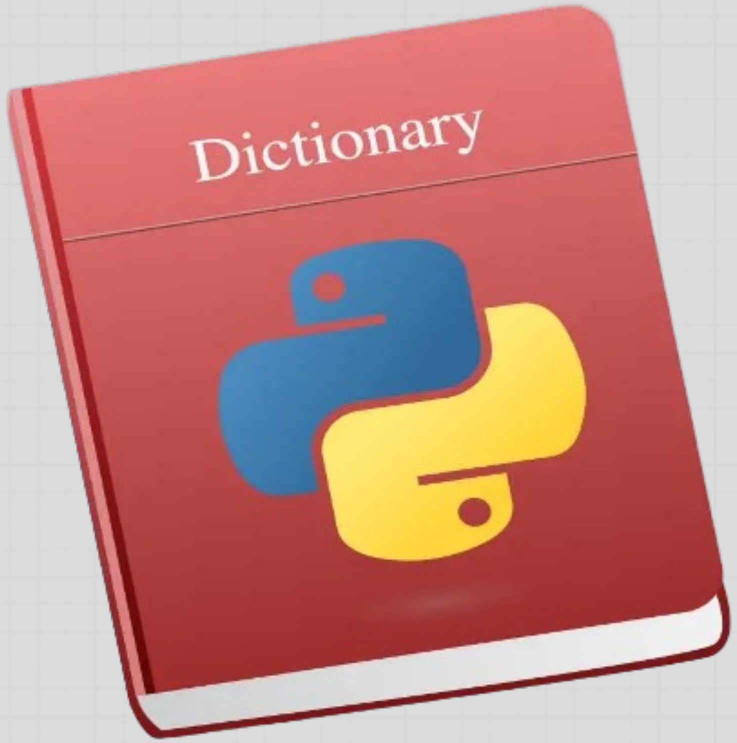
- Collection of pairs

Key	Value
name	Arthur
gender	Male
age	32
id	A01
job title	Developer
salary	110000
married	Yes



Dictionary: Key + Value

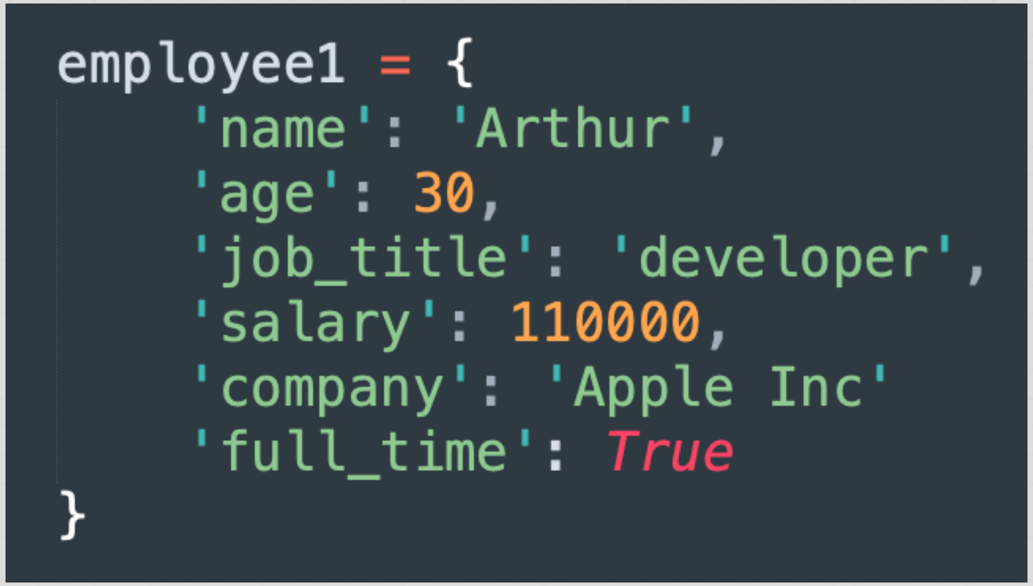
- Each value has a unique key, and we need to know the key to access the values of dictionary



Key	Value
name	Arthur
gender	Male
age	30
id	21
job_title	Developer
salary	$110000
married	False

Creating Dictionary

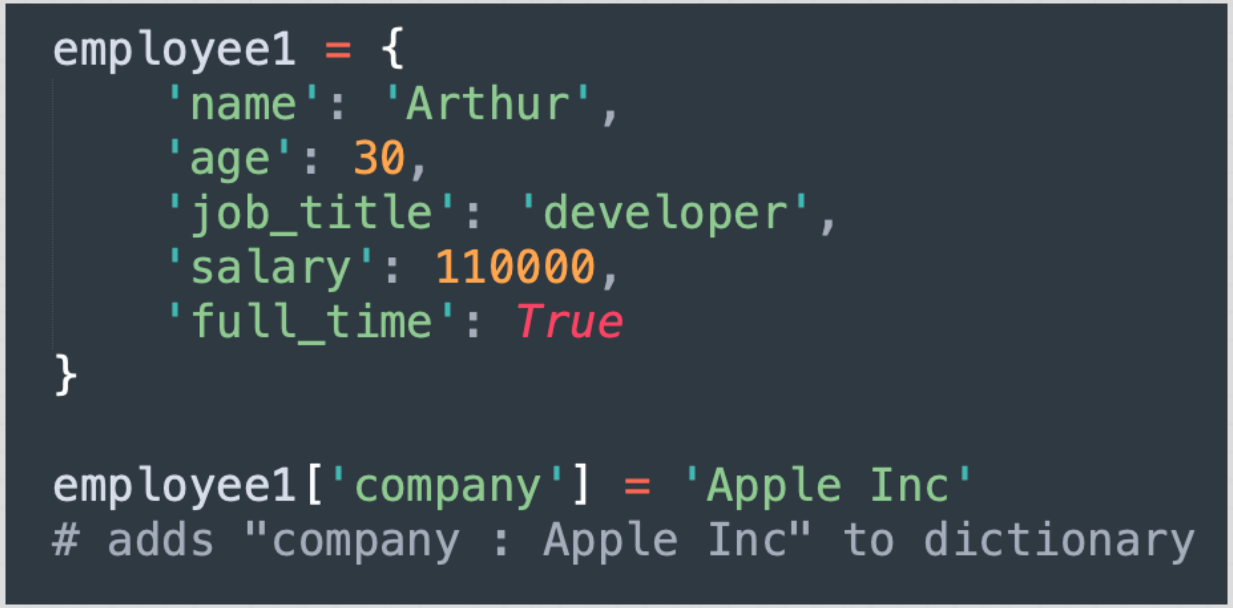
- Created by placing all the pairs inside curly brackets {key : value} separated by commas
- Items in the dictionary are ordered, changeable, and can be of any data type
- Keys in the dictionary can not be duplicated



```
employee1 = {
    'name': 'Arthur',
    'age': 30,
    'job_title': 'developer',
    'salary': 110000,
    'company': 'Apple Inc'
    'full_time': True
}
```

Adding Pairs

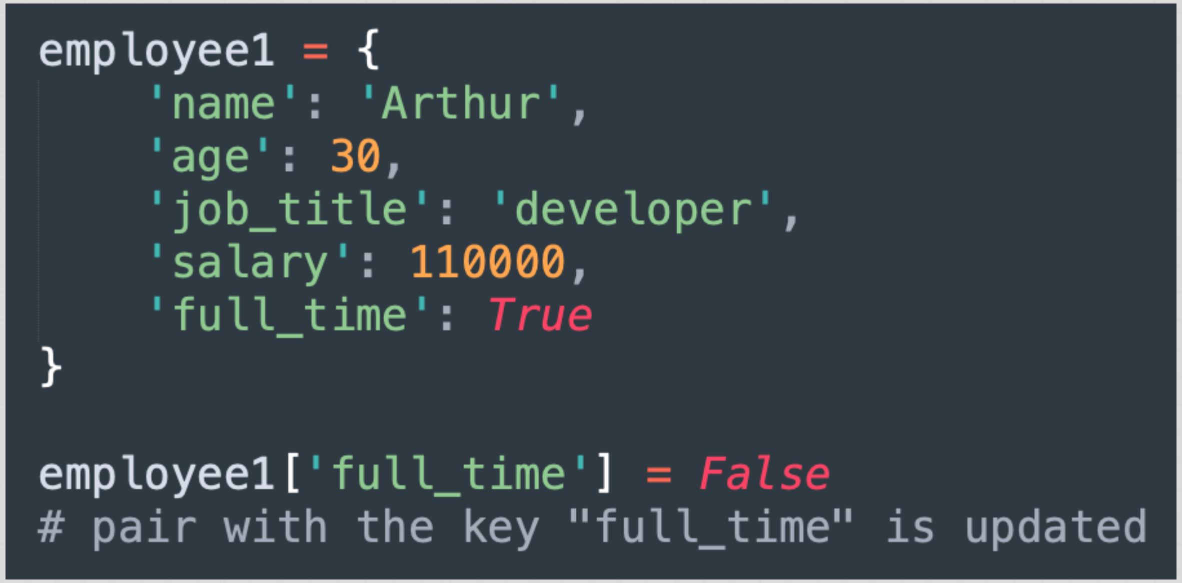
- After the dictionary is created, we can choose to add extra pairs to increase the size
- To add a pair into the dictionary, we give new index key by using square brackets and assign a value to it. [new key] = value



```python
employee1 = {
    'name': 'Arthur',
    'age': 30,
    'job_title': 'developer',
    'salary': 110000,
    'full_time': True
}

employee1['company'] = 'Apple Inc'
# adds "company : Apple Inc" to dictionary
```

Updating Pairs

- Pairs in dictionaries are changeable and we can change the value of any pair
- To change the value of a pair in the dictionary, we give index key of the pair by using square brackets and assign the new value to it. [key] = new value



```
employee1 = {
    'name': 'Arthur',
    'age': 30,
    'job_title': 'developer',
    'salary': 110000,
    'full_time': True
}

employee1['full_time'] = False
# pair with the key "full_time" is updated
```

Dictionary Methods

Method Name	Method Name	Method Name
get()	update()	pop()
popitem()	clear()	copy()
keys()	values()	items()

Nested Dictionary

- The value of a specific key in the dictionary can be a tuple, list, set, or a dictionary

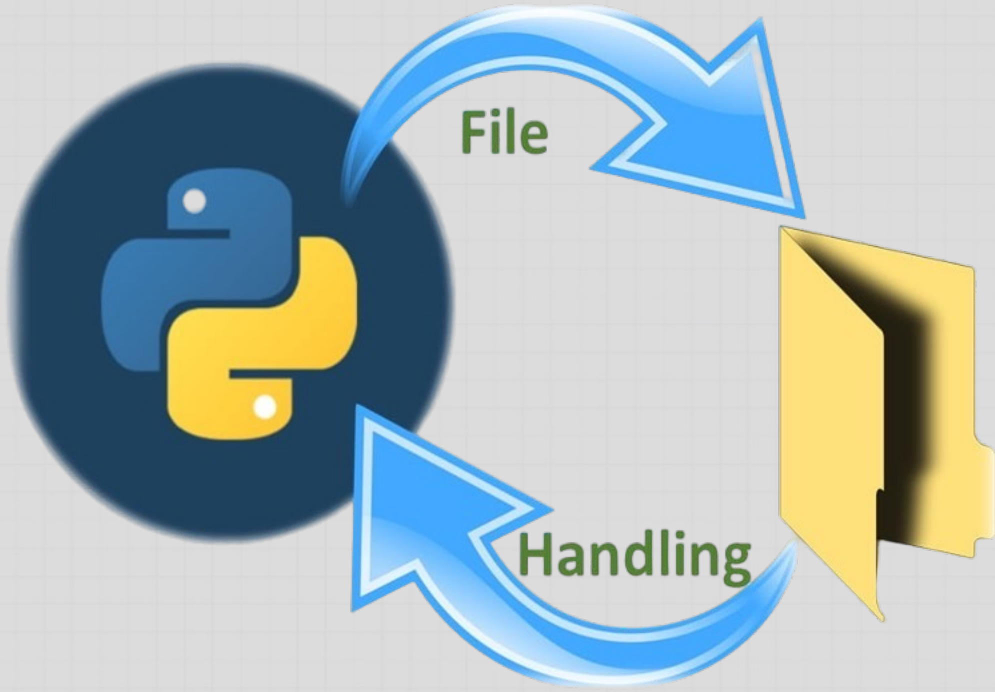
```python
employees = {
    "A01": {
        'name': 'Yulia',
        'job_title': 'Software Developer',
        'salary': 100_000,
        'full_time': True
    },

    'A02': {
        'name': 'Daniel',
        'job_title': 'Data Analyst',
        'salary': 90_000,
        'full_time': False
    }
}
```

```python
students = {
    "A01": {
        'name': 'James',
        'full_time': True,
        'gpa': 3.5
        'subjects': ['Mathematics', 'Physics']
    },

    'A02': {
        'name': 'James',
        'full_time': True,
        'gpa': 3.5
        'subjects': ['Biology', 'Chemistry']
    }
}
```

Open Files

- The build-in method open() is used for file handlings. Returns file object
- The method takes two arguments:

  open(file_path , mode)
- File handling is decided based on the second argument (mode) that's passed to the method



Open File Modes

- Syntax: Open(file_path , mode)

Modes	Descriptions
“r”	Read. Used to open a file for reading. Gives Error if the file does not exist
“w”	Write. Used to open a file for Writing. Creates the file if the file does not exist
“a”	Append. Used to open a file for appending Creates the file if the file does not exist
“x"	Create. Used to create a file. Given error if the file was already created

Read Files

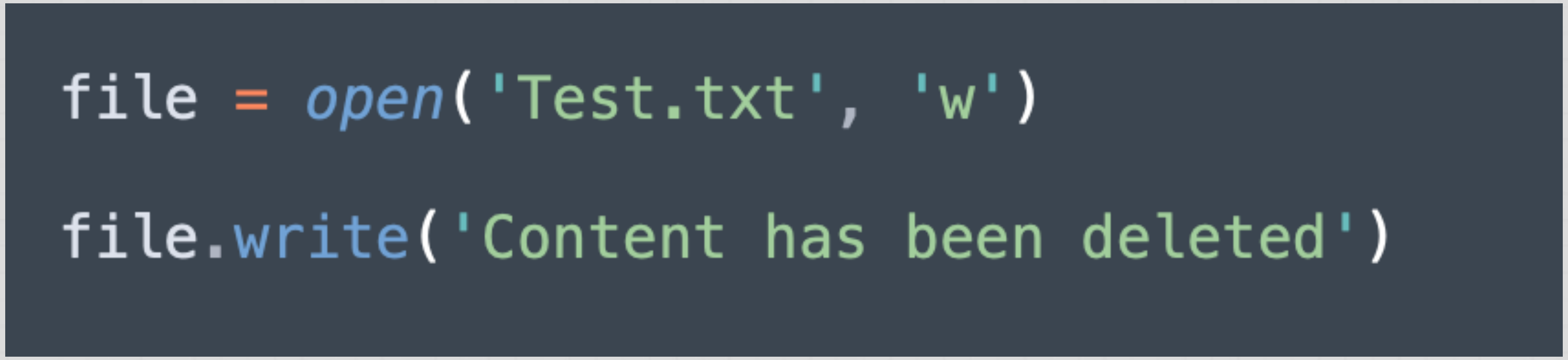
- After the build-in method open() returned file object, we can call read() method for reading the content of the file

```
file = open('Test.txt', 'r')

text = file.read()
```

```
file = open('Test.txt', 'r')

firstLine = file.readline()
secondLine = file.readline()
```

Write Files

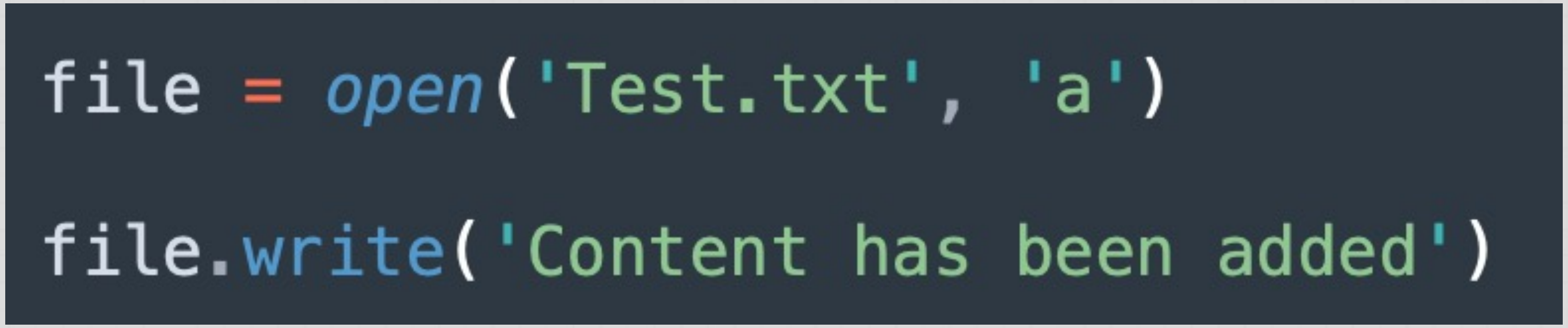
- After the build-in method open() returned file object, we can call write() method for overwriting the content of the file



```python
file = open('Test.txt', 'w')

file.write('Content has been deleted')
```

Appending

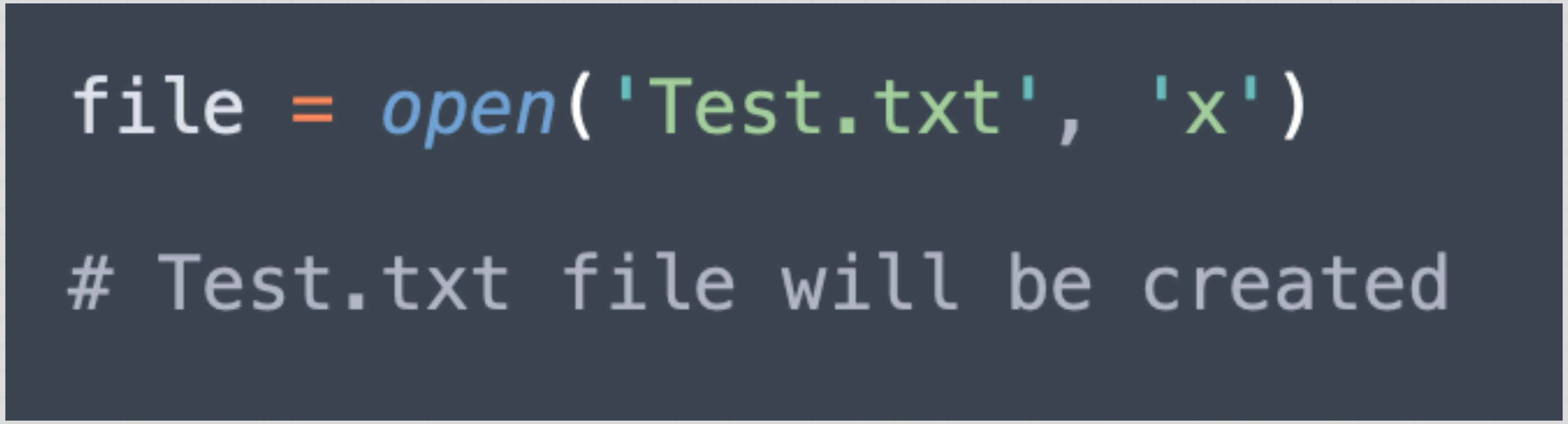
- After the build-in method open() returned file object, we can call write() method for appending to the end of the file



```
file = open('Test.txt', 'a')

file.write('Content has been added')
```

Create Files

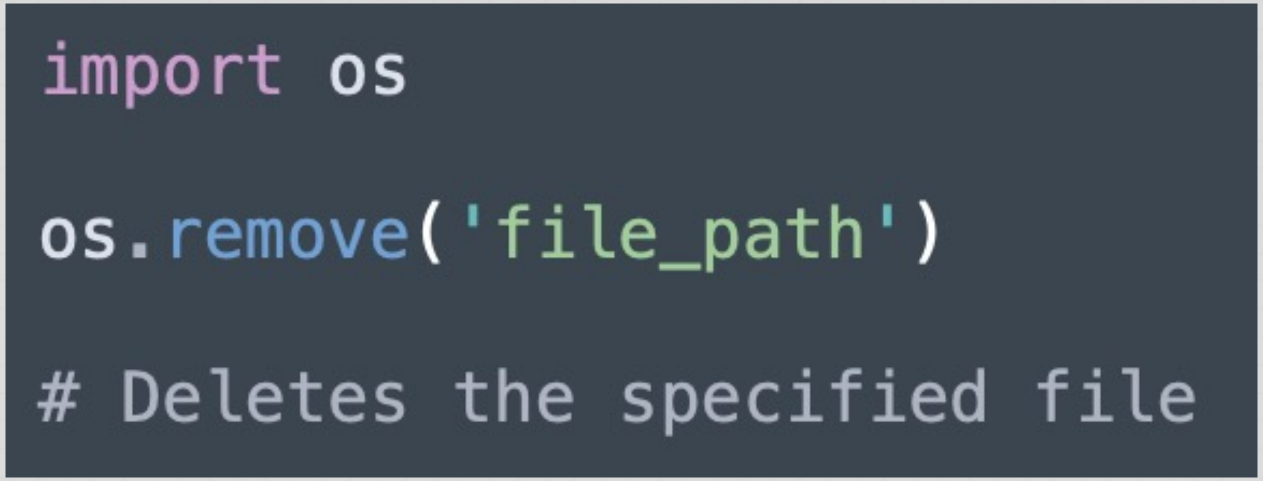
- After the build-in method open() returned file object, it creates the specified file in the specified directory



```python
file = open('Test.txt', 'x')

# Test.txt file will be created
```

Delete Files

- The OS module needs to be imported to delete a file, then we can use remove() method to remove the file



```
import os

os.remove('file_path')

# Deletes the specified file
```