



# **Object Oriented Programming**

Day 1





# **Programmer Tools**

- Text editor
- Command line
- Web Browser





## **Environment Variable**

echo \$PATH





## **Python**

- Multi-platform
- Interpreted
- Dynamic Typing
- Garbage Collected
- Object-Oriented & Procedural
- Large standard library

Created by Guido van Rossum, first released in 1991.

https://en.wikipedia.org/wiki/Python\_(programming\_language)











# **Pipenv**

Python dev workflow for Humans

https://docs.pipenv.org/





## Starting a new project

```
mkdir new_project && cd $_
pipenv --python 3.7
```

```
pipenv install nose --dev
pipenv install flask
```





```
# Pipfile
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"
[packages]
flask = "*"
[dev-packages]
nose = "*"
pylint = "*"
[requires]
python_version = "3.7"
```





## Running your code

```
pipenv shell
python file.py
exit # To quit the current virtual env
```

Or

```
pipenv run python file.py
```





#### **REPL**

```
python

# Python 3.7.0 (default, Jun 29 2018, 20:13:13)
# [Clang 9.1.0 (clang-902.0.39.2)] on darwin
# Type "help", "copyright", "credits" or "license" for models)
>>> quit()
```





# **Types & Variables**





## **Built-in Types**

```
type(None)
type(True)
type("I am a string")
type(42)
type(3.14)
type(["I am a string", 42, 3.14])
type(["I am a string", 42, 3.14])
type(("I am a string", 42, 3.14))
type(("I am a string", 42, 3.14))
type(("john": 25, "paul": 24})
# => <class 'NoneType':
# => <class 'bool'>
# => <class 'int'>
# => <class 'float'>
type(("I am a string", 42, 3.14))
# => <class 'tuple'>
type({"john": 25, "paul": 24})
```

https://docs.python.org/3/library/stdtypes.html





### **Variables**

```
# Variable assignment statement
name = "John"

# (almost) Constants
NUMBER_OF_DAYS_IN_A_WEEK = 7
```





## **String Formatting - Interpolation**

```
first_name = "John"
last_name = "Lennon"
sentence = "Hi, my name is {} {}".format(first_name, last_
```

### Since Python 3.6:

```
sentence = f"Hi my name is {first_name} {last_name}"
```





## Type casting on String

```
type('1984')  # => <type 'str'>
int('1984')  # => 1984
type(int('1984'))  # => <type 'int'>
```





## Integer

```
# Built-in functions
abs(-2)  # => 2
max(2, 3)  # => 3
```

https://docs.python.org/3/library/functions.html





### **Float**

```
11 / 2  # => 5

11.0 / 2  # => 5.5

round(3.1415926, 2) # => 3.14
```

Math module

```
import math

math.floor(3.2)  # => 3.0
math.ceil(3.2)  # => 4.0
```





### List (Mutable sequence type)

```
beatles = [ "paul", "john", "ringo" ]

beatles.append("GEORGE") # Create
print(beatles[0]) # Read
beatles[3] = "george" # Update
del beatles[3] # Delete
```

https://docs.python.org/3/library/stdtypes.html#mutable-sequence-types



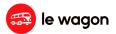


# Tuple (immutable sequence type)

```
john = ("john", "lennon", 24)
```

https://docs.python.org/3/library/stdtypes.html#tuples





https://stackoverflow.com/questions/626759/whats-the-difference-between-lists-and-tuples





## **Dictionary (Mapping Type)**

```
beatles = { "john": "guitar", "paul": "bass" }

beatles["ringo"] = "drums" # Create / Update
print(beatles["ringo"]) # Read
del beatles["ringo"] # Delete

# beatles[unknown_key] => KeyError
```

https://docs.python.org/3/library/stdtypes.html#mapping-types-dict





# **Control Flow**





## **Basic flow**

Top to bottom / line-by-line

https://docs.python.org/3/tutorial/controlflow.html





## if statement

```
if condition:
    # code executed only when condition is "truthy"
elif another_condition:
    # code executed when `condition` was falsy
    # and `another_condition` truthy
else:
    # code executed if no condition was truthy
```





## **Ternary operator**

Since Python 2.5

code\_when\_truthy if condition else code\_when\_falsey





# **Boolean logic**

#### Combinations:

```
and or not
```

### Comparisons:

```
is
is not
in
not in
<
> >
==
!=
```





## **Functions**

https://docs.python.org/3/tutorial/controlflow.html#defining-functions





```
def vote(age):
    if age < 18:
        return "You can't vote"
    else:
        return "You can vote"

print(vote(24))
# => "You can vote
```

#### Useful for:

- Don't Repeat Yourself (DRY)
- Refactoring (keep functions short)





### Parameter vs Arguments

```
def is_even(number): # `number` is a parameter
    return number % 2 == 0
```

We call a function passing arguments

```
is_even(4) # `4` is an argument
```





#### Scope

```
def greet(first_name, last_name):
    full_name = f"{first_name.capitalize()} {last_name.upg
    return f"Hello, {full_name}"

print(greet("ringo", "starr"))
# => Hello, Ringo STARR

full_name
# => NameError: name 'full_name' is not defined
```

https://docs.python.org/3/reference/executionmodel.html#resolution-of-names





# Loops





### The while statement

```
while condition:
    # executed while `condition` is truthy
    # or until reaching a `break`
```





### The for statement

```
for letter in "python":
    print(letter)

for key in {"x": 1, "y": 2}:
    print(key)

for i in range(4):
    print(i)
```

List comprehensions:

```
[x * 2 for x in range(1, 8)] # => [2, 4, 6, 8, 10, 12, 14]
```

https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions





# Classes (OOP)





### Data + Behavior

Exemple of the built-in type list:

```
# Storing data through state
beatles = [ "john", "paul" ]

# Modify its state through methods
beatles append("ringo")
```





# A Class is like a car factory







# A first Dog class

```
# dog py
class Dog():
   pass
```

- Convention: filename is in lower snake case, and class name in upper camel case
- For example: sports\_car.py => SportsCar





#### Initialization

```
scooby = Dog()
pongo = Dog()
```

We just created two new instances

```
class Dog():
    def __init__(self):
        print("d ")
    pass
```





#### Instance variable

```
class Dog():
    def __init__(self, name):
        self.name = name

scooby = Dog("Scooby")
scooby.name # => "Scooby"
```





#### Instance method

```
class Dog():
    def __init__(self, name):
        self.name = name
        self.tricks = []
    def learn(self, trick):
        self.tricks.append(trick)
pongo = Dog("Pongo")
pongo.learn("roll over")
pongo.learn("play dead")
pongo.tricks # => ['roll over', 'play dead']
```





#### **SUMMARY**

- Everything in python is an object
- OOP is about data (or state) and behavior
- State is stored in instance variables ( self.\*)
- Behavior is defined by instance methods ( def \* )





#### Inheritance

- Some classes may **share** some behavior and state...
- ... still having some **specific** behavior

```
class Dog():
    pass

class Cat():
    pass
```





### Shared/Specific

```
class Dog():
    def __init__(self, name):
        self_name = name
    def talk(self):
        return "Woof"
class Cat():
    def __init__(self, name):
        self.name = name
    def talk(self):
        return "Meow"
```





#### Inheritance

```
class Animal():
    def __init__(self, name):
        self.name = name
```

```
class Dog(Animal):
    def talk(self):
        return "Woof"

class Cat(Animal):
    def talk(self):
        return "Meow"
```





## Polymorphism

```
pongo = Dog("Pongo")
oliver = Cat("Oliver")

animals = [ pongo, oliver ]

for animal in animals:
    print(animal.talk())

# => Woof \n Meow
```





### More OOP concepts

- Python has **multiple** inheritance
- Static methods with @staticmethod decorator
- super() in an inheritance context

Python supports Abstract Base Classes. Because of dynamic typing, there is no **Interface** concept in Python.





# **Modules & Packages**





## import stuff from Modules

```
# greet.py
from sys import argv

def main():
    print(f"Hello {argv[1].capitalize()}")

if __name__ == '__main__':
    main()
```

```
python greet.py paul
# => Hello Paul
```





## Your own package

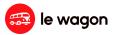
```
mkdir sound && touch sound/__init__.py
# sound/__init__.py
def play():
    return "Playing..."
# program.py
from sound import play
if __name__ == '__main__':
    print(play())
python program.py
# => Playing...
```





More at https://docs.python.org/3/tutorial/modules.html





# **Happy OOP-ing!**