
Sprint 05
Marathon Python
May 27, 2021
ucode

Contents

Engage 2
Investigate 3
Act Basic: Task 00 > Guard 5
Act Basic: Task 01 > Knight 8
Act Basic: Task 02 > Shipments 11
Act Basic: Task 03 > Decorator 17
Act Basic: Task 04 > Inheritance 20
Act Basic: Task 05 > Threads 24
Act Basic: Task 06 > Processes 26
Act Advanced: Task 07 > Dynamic 28
Act Advanced: Task 08 > New 32
Act Advanced: Task 09 > With 35
Act Advanced: Task 10 > Traffic light 38
Act Advanced: Task 11 > Bounce 40
Share 44

ucode

Engage

DESCRIPTION

Welcome to the last Sprint!

Welcome to the last Sprint!

In order to develop a proper OOP (object-oriented programming) program, it's important to understand the main OOP concepts and how they work together. The object-oriented paradigm is a completely different way of thinking in software development.

OOP was invented as an attempt to project real-world objects into program code. It was thought that objects are easier to perceive and read for the developer, because the world is easier to perceive as a set of interacting objects. This helps to design a sufficiently clear software architecture. It will be easier for people to understand, expand, modify and also test such a project. It will save a lot of time in the future.

As with anything, there are some downsides to OOP. It takes a long time to design the architecture of a good OOP program. OOP architecture does not fit all projects, it depends on scale, desired functionality, etc. So, it's important to understand the advantages and disadvantages of OOP before using it in your project.

Another advanced programming topic you will be learning is parallel programming.
The main principle of parallel computing is to divide a task into smaller parts that don't depend on each other, where each part is performed simultaneously on a separate device and then the results are connected back together. This allows to speed up completion of complex tasks, and essentially makes our devices faster and more efficient.

Let's get right into it!

BIG IDEA

Object-oriented programming.

ESSENTIAL QUESTION

What are the benefits of using OOP in software architecture?

CHALLENGE

Learn the main features of OOP.

ucode

Sprint 05 | Marathon Python **> 2**

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students around you and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- What is object-oriented programming (OOP)?
- What is the use of OOP in general?
- What is the use of OOP in Python?
- What are the benefits of OOP compared to functional programming?

- What is a class?
- What is meant by `self` in Python?
- What is the difference between a method and a function?
- What are class arguments?
- What are `threads`?
- What are `processes`?
- What is the difference between a process and a thread?

GUIDING ACTIVITIES

Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

- Watch this video about OOP. Find more resources.
- Read about Python classes.
- Watch a video about how CPU works with data.
- Look at this video about the difference between processes and threads.
- Read this article on Multithreading vs Multiprocessing in Python.
- Clone your git repository issued on the challenge page in the LMS.
- Proceed with tasks.

ANALYSIS

Analyze your findings. What conclusions have you made after completing guiding questions and activities? In addition to your thoughts and conclusions, here are some more analysis results.

- Be attentive to all statements of the story.
- All tasks are divided into Act Basic and Act Advanced. You need to complete all basic tasks to validate the Sprint. But to achieve maximum points, consider accomplishing advanced tasks also.
- Analyze all information you have collected during the preparation stages. Try to define the order of your actions.

Sprint 05 | Marathon Python **> 3**

- Perform only those tasks that are given in this document.
- Submit only those files that are described in the story. Only useful files allowed, garbage shall not pass!
- Run the scripts using `python3`.
- Make sure that you have `Python` with a `3.8` version, or higher.
- Use the standard library available after installing `Python`. You may use additional packages/libraries that were not previously installed only if they are specified in the task.
- To figure out what went wrong in your code, use PEP 553 -- Built-in breakpoint().
- Complete tasks according to the rules specified in the PEP8 conventions.
- The solution will be checked and graded by students like you. Peer-to-Peer learning.
- Also, the challenge will pass automatic evaluation which is called `Oracle`.
- If you have any questions or don't understand something, ask other students or just Google it.

Act Basic: Task 00

NAME

Guard

DIRECTORY

`t00_guard/`

SUBMIT

`guard.py`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- How to define a class in Python?
- What are classes used for?
- What can a class contain?
- What is an instance of a class? How is an instance different from a class?
- How to create an instance of a class?
- What does the method `__init__()` do?
- How is `__init__()` called?

Let's try to create a simple class. Open the PYTHON INTERPRETER and repeat the actions.

```
>python3
>>> # define a Person class containing a method 'greet()' that prints a message
>>> class Person:
...     def greet(self):
...         print('Hello!')
```

```
...
>>> # create an instance of the Person class
>>> person = Person()
>>> # call the method 'greet()' for the created instance
>>> person.greet()
Hello!
>>> quit()
>
```

Now let's try using the initializer.

```
>python3
>>> # define a Person class containing '__init__()'
>>> # and a method 'greet()' that prints a message
>>> class Person:
...     def __init__(self, name):
...         self.name = name
...
...     def greet(self):
...         print(f'My name is {self.name}!\nGlad to welcome you!')
```

Sprint 05 | Marathon Python **> 5**

```
...
>>> person = Person('John')
>>> person.name
'John'
>>> person.greet()
My name is John!
Glad to welcome you!
>>>
```

DESCRIPTION

Create a class `Guard` that:

- has an initializer `__init__()` that takes a dynamic number of named parameters `**kwargs`.
  Parameters `name` and `salary` must be set in the `__init__()`.
  If the name is not passed, set it to None.
  If the salary is not passed, set it to 0
- has two methods:
  - `greet()` prints to the console:
    'Hello, my name is [name]!'
  - `receive_bribe()` takes a number and prints a message. If the amount passed is greater than the salary of the guard, print:
    'You may pass.'
    Otherwise, print:
    'I am not letting you pass.'

EXAMPLE

```
from guard import Guard


if __name__ == "__main__":
    print(f'***EXAMPLE 1***')
    guard = Guard(name='Christopher')
    print(guard.greet())
    print(guard.receive_bribe(10))

    print(f'***EXAMPLE 2***')
    guard = Guard(salary=100)
```

```
print(guard.greet())
print(guard.receive_bribe(95))
print(guard.receive_bribe(105))

print(f'***EXAMPLE 3***')
guard = Guard(name='Christopher', salary=100)
print(guard.greet())
print(guard.receive_bribe(80))
print(guard.receive_bribe(135))
```

CONSOLE VIEW

```
>python3 s05t00_guard_main.py
***EXAMPLE 1***
Hello, my name is Christopher!
You may pass.
***EXAMPLE 2***
Hello, my name is None!
I am not letting you pass.
You may pass.
***EXAMPLE 3***
Hello, my name is Christopher!
I am not letting you pass.
You may pass.
>
```

SEE ALSO

Python Classes and Objects
The Python Tutorial - Classes
Python - Object Oriented

Act Basic: Task 01

NAME

Knight

DIRECTORY

`t01_knight/`

SUBMIT

`knight.py`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What are the different ways of working with `**kwargs` in class initialization?
- How to set an attribute of an object using a method?
- How to check whether an object has a certain attribute using a method?

LEGEND

The wise Sir Bedevere was the first to join King Arthur's knights, but other illustrious names were soon to follow: Sir Lancelot the Brave; Sir Galahad the Pure; and Sir Robin the Not-quite-so-brave-as-Sir-Lancelot who had nearly fought the Dragon of Angnor, who had nearly stood up to the vicious Chicken of Bristol and who had personally wet himself at the Battle of Badon Hill; and the aptly named Sir Not-appearing-in-this-film. Together they formed a band whose names and deeds were to be retold throughout the centuries, the Knights of the Round Table.

-- Monty Python and the Holy Grail

DESCRIPTION

Create a class `Knight`. The class has an initializer that can be called with any number of named parameters.

Also, it has a method `greet()` that prints a message. If the instance of the class has the attributes `'name'` and `'title'` (check using the method `hasattr()`), the message is `"Hello, I'm Sir [name] the [title]!"`. Otherwise, the message is `'Hello!'`.

The script in the EXAMPLE tests your function. If everything is correct, it should generate output as seen in the CONSOLE VIEW. Also, you can see examples in the PYTHON INTERPRETER. Pay attention that you must only submit the file knight.py, not the test script.

EXAMPLE

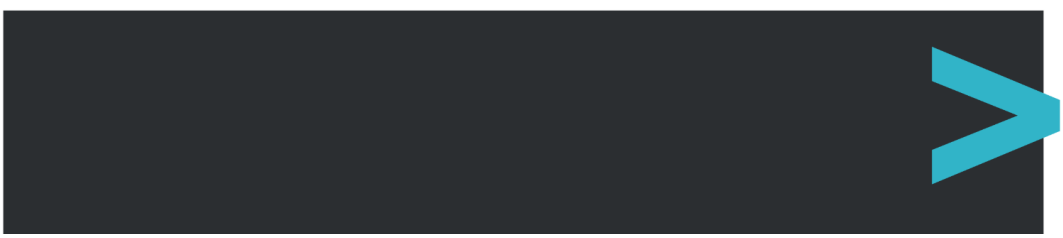
```python
import json
from knight import Knight


if __name__ == '__main__':
    knights = [
        Knight(
            name='Robin',
            title='Not-quite-so-brave-as-Sir-Lancelot',
            deeds=['Nearly fought the Dragon of Angnor.',
                   'Nearly stood up to the vicious Chicken of Bristol.',
                   'Personally wet himself at the Battle of Badon Hill.'],
            group='Knights of the Round Table',
            gender='male'),
        Knight(name='Bedevere'),
        Knight(eyes='brown', age=30)
    ]

    for knight in knights:
        print(json.dumps(knight.__dict__, indent=1))
        knight.greet()
        print('---')
```

CONSOLE VIEW

```
>python3 s05t01_knight_main.py
{
 "name": "Robin",
 "title": "Not-quite-so-brave-as-Sir-Lancelot",
 "deeds": [
  "Nearly fought the Dragon of Angnor.",
  "Nearly stood up to the vicious Chicken of Bristol.",
  "Personally wet himself at the Battle of Badon Hill."
 ],
 "group": "Knights of the Round Table",
 "gender": "male"
}
Hello, I'm Sir Robin the Not-quite-so-brave-as-Sir-Lancelot!
---
{
 "name": "Bedevere"
}
Hello!
---
{
 "eyes": "brown",
 "age": 30
```

ucode

Sprint 05 | Marathon Python **> 9**



```
}
Hello!
---
>
```

PYTHON INTERPRETER

```
>python3
>>> from knight import Knight
>>> lancelot = Knight(name='Lancelot', title='Brave',
...                                 age=40)
>>> lancelot
<knight.Knight object at 0x7efe3c1d5370>
>>> lancelot.__dict__
{'name': 'Lancelot', 'title': 'Brave', 'age': 40}
>>> lancelot.greet()
Hello, I'm Sir Lancelot the Brave!
>>> tim = Knight(name='Tim', job='enchanter')
>>> tim.__dict__
{'name': 'Tim', 'job': 'enchanter'}
>>> tim.greet()
Hello!
>>> knight = Knight(nickname='Bob')
>>> knight.__dict__
{'nickname': 'Bob'}
>>> knight.greet()
Hello!
>>>
```

SEE ALSO

Python setattr()
Python hasattr()
10 Examples to Master *args and **kwargs in Python

ucode

Sprint 05 | Marathon Python **> 10**

Act Basic: Task 02

NAME

Shipments

DIRECTORY

`t02_shipments/`

SUBMIT

shipments.py

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What relationships may exist between classes in OOP?
- What is aggregation in OOP?
- What is composition in OOP?
- What is the difference between aggregation and composition?
- What is the magic `__str__()` method used for?
- What is the difference between `__str__()` and `__repr__()`?
- How to redirect the log output to a file?

DESCRIPTION

Create a script that contains three classes: `Cargo`, `Container`, and `Ship`. They are connected in the following way: a ship can have many containers; every container can have one cargo.

Here are more details on the classes:

- `Cargo` class must have
  - `destination` property (string)
  - `weight` property (integer)
  - an initializer that sets the given `destination` and `weight`
- `Container` class must have
  - `weight_limit` property (integer)
  - `cargo` property (`Cargo` instance)
  - an initializer that sets the given `weight_limit` and `cargo` (default is `None`)
  - `set_cargo()` method that sets a passed `Cargo` instance to the `cargo` property if its weight is less or equal to the `weight_limit`
- `Ship` class must have

Sprint 05 | Marathon Python **> 11**

  - `route` property (list of strings)
  - `containers` property (list of `Container` instances)
  - an initializer that sets the given `route` and `containers`
  - `add_containers()` method that takes a list of `Container` instances, loops through it, and adds to the `containers` property those that have a `cargo` with a `destination` on the ship's `route`

For each of the classes, add an override for the `__str__()` and the `__repr__()` methods. See the exact formatting in the PYTHON INTERPRETER and CONSOLE VIEW sections.

Using the logging Python module, log to a shipments.log file. It must be overwritten, not appended to.

Use the s05t02_shipments_main.py file from resources to test your solution. Your output must look like the CONSOLE VIEW.

EXAMPLE

```python
import random
import sys
from shipments import Cargo, Container, Ship


def to_set():
    return random.randint(0, 1)


if __name__ == '__main__':
    random.seed(sys.argv[1])  # set seed in command line
    destinations = ['Tianjin', 'Antwerp', 'Los Angeles', 'Xiamen', 'Bremen',
                    'Santos', 'Barcelona']
    containers = []
    # creating random cargo and containers
    for i in range(16):
        destination = random.choice(destinations)
        weight = random.randint(1000, 3000)
        weight_limit = random.randint(1000, 4000)
        cargo = Cargo(destination, weight)
        print(cargo)
        # setting cargo either in __init__ or in method directly
        if to_set():
            container = Container(weight_limit, cargo)
        else:
            container = Container(weight_limit)
            container.set_cargo(cargo)
        print(container)
        containers.append(container)

    # creating a ship with a random route and the created containers
```

ucode

Sprint 05 | Marathon Python **> 12**

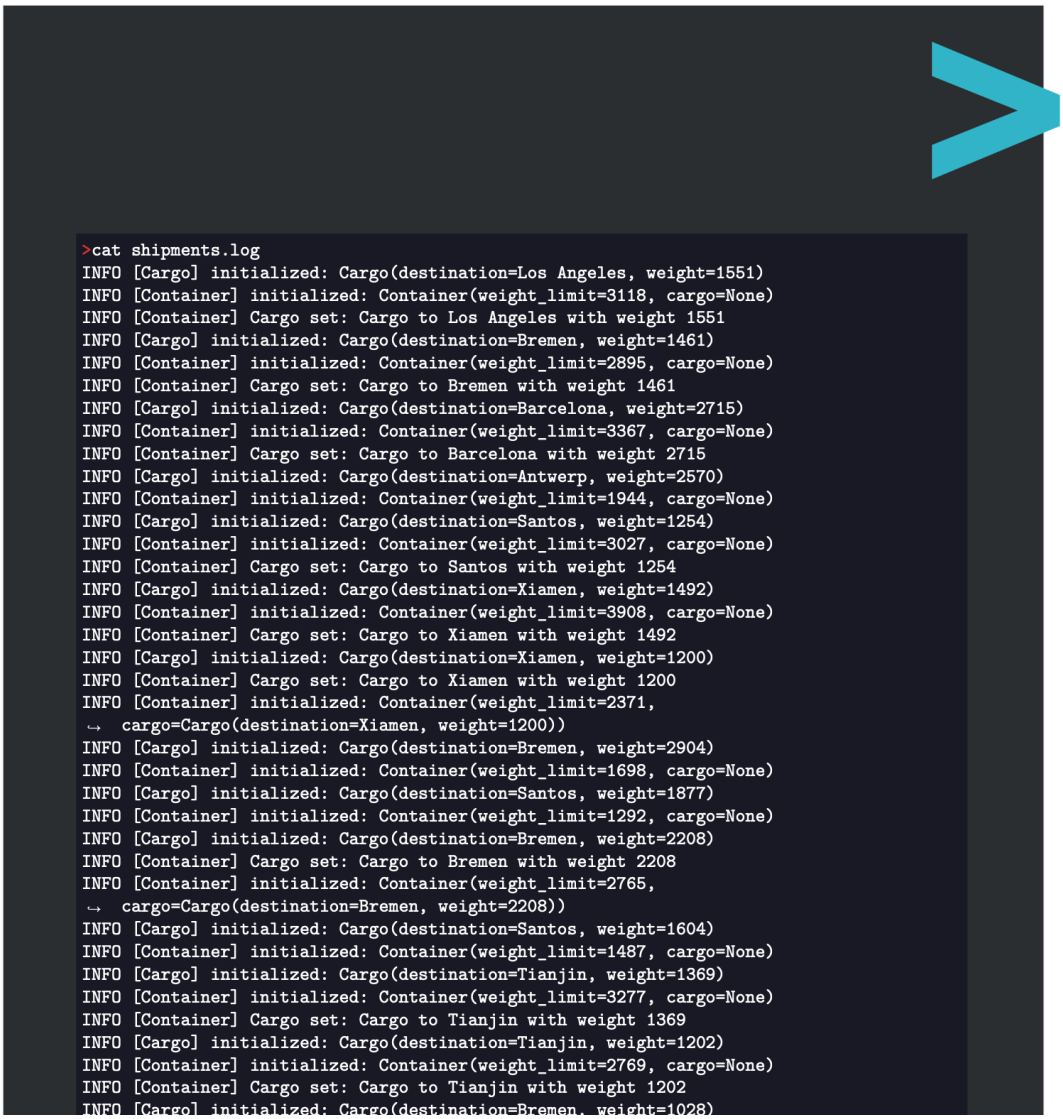
```python
    route = random.sample(destinations, 3)
    # adding part of the containers in the __init__
    ship = Ship(route, containers[:8])
    # part with method directly
    ship.add_containers(containers[8:])
    print(ship)
```

CONSOLE VIEW

```
>python3 s05t02_shipments_main.py 5
Cargo to Los Angeles with weight 1551
Container up to 3118 with Cargo to Los Angeles with weight 1551
Cargo to Bremen with weight 1461
Container up to 2895 with Cargo to Bremen with weight 1461
Cargo to Barcelona with weight 2715
Container up to 3367 with Cargo to Barcelona with weight 2715
Cargo to Antwerp with weight 2570
Container up to 1944 with None
Cargo to Santos with weight 1254
Container up to 3027 with Cargo to Santos with weight 1254
Cargo to Xiamen with weight 1492
Container up to 3908 with Cargo to Xiamen with weight 1492
Cargo to Xiamen with weight 1200
Container up to 2371 with Cargo to Xiamen with weight 1200
Cargo to Bremen with weight 2904
Container up to 1698 with None
Cargo to Santos with weight 1877
```

```
Container up to 1292 with None
Cargo to Bremen with weight 2208
Container up to 2765 with Cargo to Bremen with weight 2208
Cargo to Santos with weight 1604
Container up to 1487 with None
Cargo to Tianjin with weight 1369
Container up to 3277 with Cargo to Tianjin with weight 1369
Cargo to Tianjin with weight 1202
Container up to 2769 with Cargo to Tianjin with weight 1202
Cargo to Bremen with weight 1028
Container up to 3945 with Cargo to Bremen with weight 1028
Cargo to Xiamen with weight 1353
Container up to 1044 with None
Cargo to Santos with weight 2334
Container up to 3814 with Cargo to Santos with weight 2334
Ship to ['Barcelona', 'Xiamen', 'Tianjin'] with containers:
Container up to 3367 with Cargo to Barcelona with weight 2715
Container up to 3908 with Cargo to Xiamen with weight 1492
Container up to 2371 with Cargo to Xiamen with weight 1200
Container up to 3277 with Cargo to Tianjin with weight 1369
Container up to 2769 with Cargo to Tianjin with weight 1202
```

Sprint 05 | Marathon Python **> 13**



```
>cat shipments.log
INFO [Cargo] initialized: Cargo(destination=Los Angeles, weight=1551)
INFO [Container] initialized: Container(weight_limit=3118, cargo=None)
INFO [Container] Cargo set: Cargo to Los Angeles with weight 1551
INFO [Cargo] initialized: Cargo(destination=Bremen, weight=1461)
INFO [Container] initialized: Container(weight_limit=2895, cargo=None)
INFO [Container] Cargo set: Cargo to Bremen with weight 1461
INFO [Cargo] initialized: Cargo(destination=Barcelona, weight=2715)
INFO [Container] initialized: Container(weight_limit=3367, cargo=None)
INFO [Container] Cargo set: Cargo to Barcelona with weight 2715
INFO [Cargo] initialized: Cargo(destination=Antwerp, weight=2570)
INFO [Container] initialized: Container(weight_limit=1944, cargo=None)
INFO [Cargo] initialized: Cargo(destination=Santos, weight=1254)
INFO [Container] initialized: Container(weight_limit=3027, cargo=None)
INFO [Container] Cargo set: Cargo to Santos with weight 1254
INFO [Cargo] initialized: Cargo(destination=Xiamen, weight=1492)
INFO [Container] initialized: Container(weight_limit=3908, cargo=None)
INFO [Container] Cargo set: Cargo to Xiamen with weight 1492
INFO [Cargo] initialized: Cargo(destination=Xiamen, weight=1200)
INFO [Container] Cargo set: Cargo to Xiamen with weight 1200
INFO [Container] initialized: Container(weight_limit=2371,
↪  cargo=Cargo(destination=Xiamen, weight=1200))
INFO [Cargo] initialized: Cargo(destination=Bremen, weight=2904)
INFO [Container] initialized: Container(weight_limit=1698, cargo=None)
INFO [Cargo] initialized: Cargo(destination=Santos, weight=1877)
INFO [Container] initialized: Container(weight_limit=1292, cargo=None)
INFO [Cargo] initialized: Cargo(destination=Bremen, weight=2208)
INFO [Container] Cargo set: Cargo to Bremen with weight 2208
INFO [Container] initialized: Container(weight_limit=2765,
↪  cargo=Cargo(destination=Bremen, weight=2208))
INFO [Cargo] initialized: Cargo(destination=Santos, weight=1604)
INFO [Container] initialized: Container(weight_limit=1487, cargo=None)
INFO [Cargo] initialized: Cargo(destination=Tianjin, weight=1369)
INFO [Container] initialized: Container(weight_limit=3277, cargo=None)
INFO [Container] Cargo set: Cargo to Tianjin with weight 1369
INFO [Cargo] initialized: Cargo(destination=Tianjin, weight=1202)
INFO [Container] initialized: Container(weight_limit=2769, cargo=None)
INFO [Container] Cargo set: Cargo to Tianjin with weight 1202
INFO [Cargo] initialized: Cargo(destination=Bremen, weight=1028)
```

```
INFO [Container] initialized: Container(weight_limit=3945, cargo=None)
INFO [Container] Cargo set: Cargo to Bremen with weight 1028
INFO [Cargo] initialized: Cargo(destination=Xiamen, weight=1353)
INFO [Container] initialized: Container(weight_limit=1044, cargo=None)
INFO [Cargo] initialized: Cargo(destination=Santos, weight=2334)
INFO [Container] initialized: Container(weight_limit=3814, cargo=None)
INFO [Container] Cargo set: Cargo to Santos with weight 2334
INFO [Ship] Added Container: Container up to 3367 with Cargo to Barcelona with weight
↪  2715
INFO [Ship] Added Container: Container up to 3908 with Cargo to Xiamen with weight 1492
INFO [Ship] Added Container: Container up to 2371 with Cargo to Xiamen with weight 1200
```

Sprint 05 | Marathon Python **> 14**

```
INFO [Ship] initialized: Ship(route=['Barcelona', 'Xiamen', 'Tianjin'],
↪  containers=[Container(weight_limit=3367, cargo=Cargo(destination=Barcelona,
↪  weight=2715)), Container(weight_limit=3908, cargo=Cargo(destination=Xiamen,
↪  weight=1492)), Container(weight_limit=2371, cargo=Cargo(destination=Xiamen,
↪  weight=1200))])
INFO [Ship] Added Container: Container up to 3277 with Cargo to Tianjin with weight 1369
INFO [Ship] Added Container: Container up to 2769 with Cargo to Tianjin with weight 1202
>
```

PYTHON INTERPRETER

```
>python3
>>> from shipments import Cargo, Container, Ship
>>>
>>> # *** CARGO ***
>>> cargo = Cargo('New York', 1453)
>>> # __repr__ of cargo
>>> repr(cargo)
'Cargo(destination=New York, weight=1453)'
>>> cargo
Cargo(destination=New York, weight=1453)
>>> # __str__ of cargo
>>> str(cargo)
'Cargo to New York with weight 1453'
>>> print(cargo)
Cargo to New York with weight 1453
>>>
>>> *** CONTAINER ***
>>> cnt1 = Container(1804, cargo)
>>> cnt2 = Container(500, cargo)
>>> cnt3 = Container(5000)
>>> cnt4 = Container(2000)
>>> cnt4.set_cargo(Cargo('Shanghai', 1900))
>>> cnt1
Container(weight_limit=1804, cargo=Cargo(destination=New York, weight=1453))
>>> cnt2, cnt3, cnt4
(Container(weight_limit=500, cargo=None), Container(weight_limit=5000, cargo=None),
↪  Container(weight_limit=2000, cargo=Cargo(destination=Shanghai, weight=1900)))
>>> print(cnt1)
Container up to 1804 with Cargo to New York with weight 1453
>>> print(cnt2, cnt3, cnt4, sep='\n')
Container up to 500 with None
Container up to 5000 with None
Container up to 2000 with Cargo to Shanghai with weight 1900
>>>
>>> # *** SHIP ***
```

```
>>> ship
```

```
Ship(route=['Singapore', 'New York', 'Kyiv'], containers=[Container(weight_limit=1804,
↪  cargo=Cargo(destination=New York, weight=1453))])
>>> print(ship)
Ship to ['Singapore', 'New York', 'Kyiv'] with containers:
Container up to 1804 with Cargo to New York with weight 1453
>>> cnt3.set_cargo(Cargo('Singapore', 1200))
>>> cnt2.set_cargo(Cargo('Singapore', 300))
>>> ship.add_containers([cnt2, cnt3, cnt4])
>>> print(ship)
Ship to ['Singapore', 'New York', 'Kyiv'] with containers:
Container up to 1804 with Cargo to New York with weight 1453
Container up to 500 with Cargo to Singapore with weight 300
Container up to 5000 with Cargo to Singapore with weight 1200
>>>
>>> # *** LOGGING ***
>>> import os
>>> os.system('cat shipments.log')
INFO [Cargo] initialized: Cargo(destination=New York, weight=1453)
INFO [Container] Cargo set: Cargo to New York with weight 1453
INFO [Container] initialized: Container(weight_limit=1804, cargo=Cargo(destination=New
↪  York, weight=1453))
INFO [Container] initialized: Container(weight_limit=500, cargo=None)
INFO [Container] initialized: Container(weight_limit=5000, cargo=None)
INFO [Container] initialized: Container(weight_limit=2000, cargo=None)
INFO [Cargo] initialized: Cargo(destination=Shanghai, weight=1900)
INFO [Container] Cargo set: Cargo to Shanghai with weight 1900
INFO [Ship] Added Container: Container up to 1804 with Cargo to New York with weight 1453
INFO [Ship] initialized: Ship(route=['Singapore', 'New York', 'Kyiv'],
↪  containers=[Container(weight_limit=1804, cargo=Cargo(destination=New York,
↪  weight=1453))])
INFO [Cargo] initialized: Cargo(destination=Singapore, weight=1200)
INFO [Container] Cargo set: Cargo to Singapore with weight 1200
INFO [Cargo] initialized: Cargo(destination=Singapore, weight=300)
INFO [Container] Cargo set: Cargo to Singapore with weight 300
INFO [Ship] Added Container: Container up to 500 with Cargo to Singapore with weight 300
INFO [Ship] Added Container: Container up to 5000 with Cargo to Singapore with weight
↪  1200
0
>>>
```

SEE ALSO

What is the __str__ method in Python?
Aggregation vs. Composition in Object Oriented Programming

Act Basic: Task 03

NAME

Decorator

DIRECTORY

`t03_decorator/`

SUBMIT

`decorator.py`

BEFORE YOU BEGIN

In order to complete this task, you must be able to answer the following questions:

- What is a decorator?
- What does a decorator take and what does it return?
- What are decorators for?
- What kind of decorators are there?

DESCRIPTION

Create a script that uses the solution from the Task 02 Shipments. However, in this task, you will log in a different way.

Create a `log()` decorator that must print the class name and the name of the called function (in the format visible in the CONSOLE VIEW). Messages must be logged on the INFO level to the stdout.

Use the s05t03_decorator_main.py file from resources to test your solution. Your output must look like the CONSOLE VIEW.

EXAMPLE

```
from decorator import Cargo, Container, Ship

if __name__ == '__main__':
    cargos = [Cargo('Hamburg', 1200),
              Cargo('Kaohsiung', 700),
              Cargo('Hamburg', 8000),
              Cargo('Manila', 1500)]

    containers = [Container(1000, cargos[0]),
                  Container(3000, cargos[1]),
                  Container(10000, cargos[2]),
                  Container(2000)]
    containers[3].set_cargo(cargos[3])

    ship = Ship(['Manila', 'Hamburg', 'Fuzhou', 'Piraeus', 'Kaohsiung'],
                containers)
```

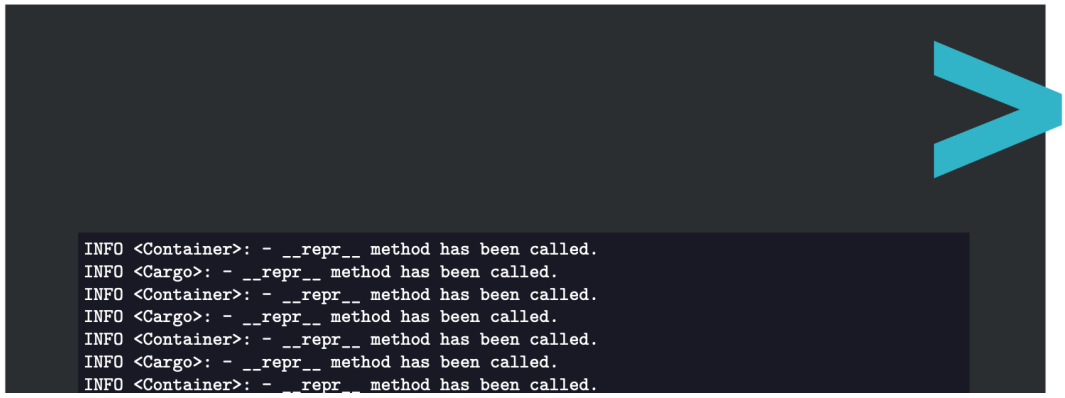
ucode

Sprint 05 | Marathon Python **> 17**

```
    ship.add_containers([Container(3000, Cargo('Hamburg', 3000))])

    print('\n*** Ship __str__ ***')
    print(ship)
    print('\n*** Ship __repr__ ***')
    print(repr(ship))
```

CONSOLE VIEW

```
>python3 s05t03_decorator_main.py
INFO <Cargo>: - __init__ method has been called.
INFO <Cargo>: - __init__ method has been called.
INFO <Cargo>: - __init__ method has been called.
INFO <Cargo>: - __init__ method has been called.
INFO <Container>: - set_cargo method has been called.
INFO <Container>: - __init__ method has been called.
INFO <Container>: - set_cargo method has been called.
INFO <Container>: - __init__ method has been called.
INFO <Container>: - set_cargo method has been called.
INFO <Container>: - __init__ method has been called.
INFO <Container>: - set_cargo method has been called.
INFO <Container>: - __init__ method has been called.
INFO <Container>: - set_cargo method has been called.
INFO <Ship>: - add_containers method has been called.
INFO <Ship>: - __init__ method has been called.
INFO <Cargo>: - __init__ method has been called.
INFO <Container>: - set_cargo method has been called.
INFO <Container>: - __init__ method has been called.
INFO <Ship>: - add_containers method has been called.

*** Ship __str__ ***
INFO <Cargo>: - __str__ method has been called.
INFO <Container>: - __str__ method has been called.
INFO <Cargo>: - __str__ method has been called.
INFO <Container>: - __str__ method has been called.
INFO <Cargo>: - __str__ method has been called.
INFO <Container>: - __str__ method has been called.
INFO <Cargo>: - __str__ method has been called.
INFO <Container>: - __str__ method has been called.
INFO <Ship>: - __str__ method has been called.
Ship to ['Manila', 'Hamburg', 'Fuzhou', 'Piraeus', 'Kaohsiung'] with containers:
Container up to 3000 with Cargo to Kaohsiung with weight 700
Container up to 10000 with Cargo to Hamburg with weight 8000
Container up to 2000 with Cargo to Manila with weight 1500
Container up to 3000 with Cargo to Hamburg with weight 3000

*** Ship __repr__ ***
INFO <Cargo>: - __repr__ method has been called.
```

```
INFO <Ship>: - __repr__ method has been called.
Ship(route=['Manila', 'Hamburg', 'Fuzhou', 'Piraeus', 'Kaohsiung'],
↪  containers=[Container(weight_limit=3000, cargo=Cargo(destination=Kaohsiung,
↪  weight=700)), Container(weight_limit=10000, cargo=Cargo(destination=Hamburg,
↪  weight=8000)), Container(weight_limit=2000, cargo=Cargo(destination=Manila,
↪  weight=1500)), Container(weight_limit=3000, cargo=Cargo(destination=Hamburg,
↪  weight=3000))])
```

PYTHON INTERPRETER

```
>python3
>>> from decorator import log
>>> # testing log decorator alone
>>> class Snake:
...     @log
...     def hiss(self):
...             print('hissss')
...
>>> snake = Snake()
>>> snake.hiss()
hissss
INFO <Snake>: - hiss method has been called.
>>>
```

SEE ALSO

PEP 318 -- Decorators for Functions and Methods
Python Decorators

Sprint 05 | Marathon Python **> 19**