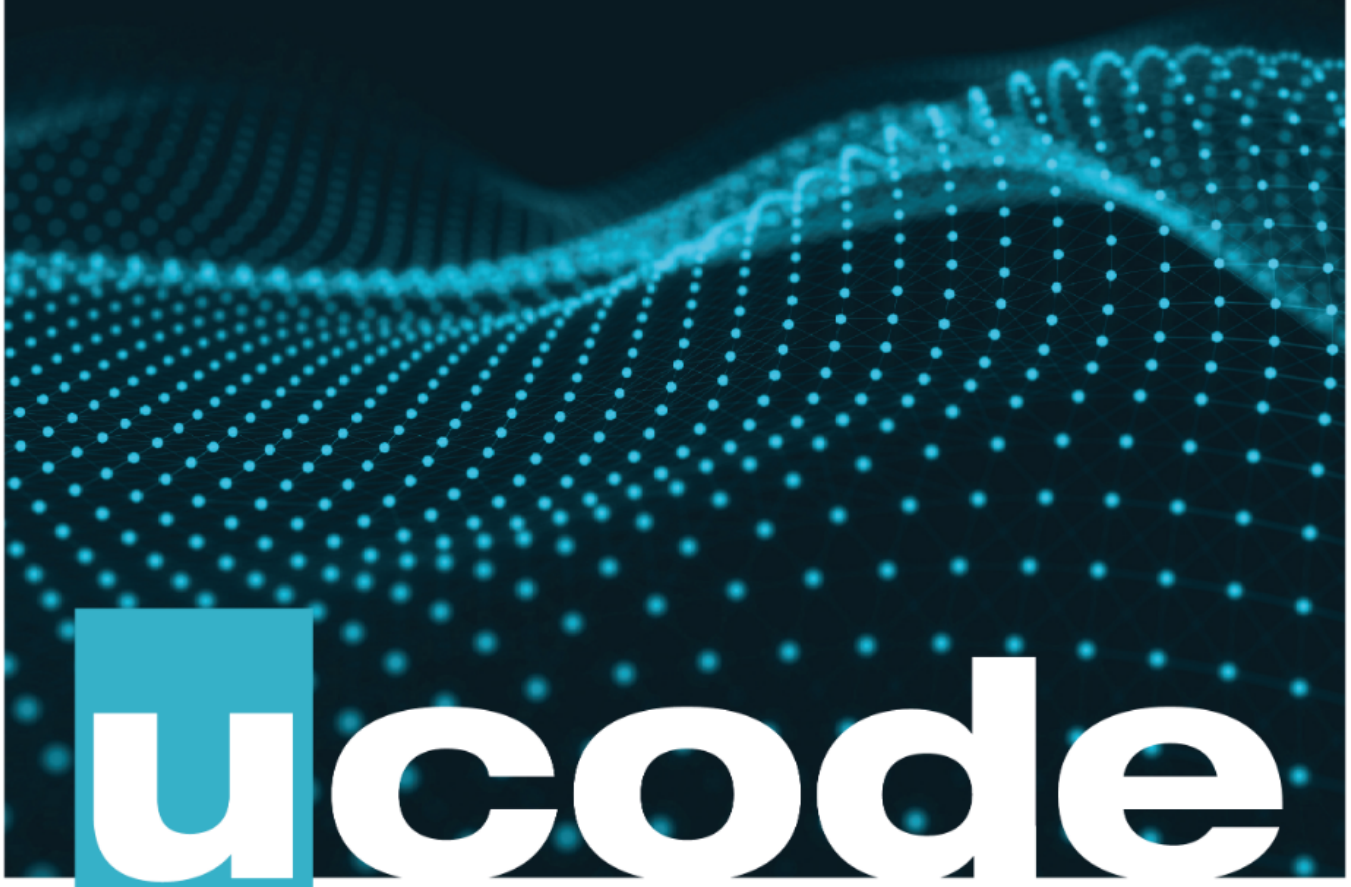
Sprint 05

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!

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.

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.

- 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!')
```

```
...
>>> 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

```python
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

```
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
```

```
}
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

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

- `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

```
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
```

```
    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
```

```
>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
```

```
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(['Singapore', 'New York', 'Kyiv'], [cnt1, cnt2, cnt3, cnt4])
>>> 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)
```

```
    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 <Container>: - __repr__ method has been called.
INFO <Cargo>: - __repr__ method has been called.
INFO <Container>: - __repr__ method has been called.
INFO <Cargo>: - __repr__ method has been called.
INFO <Container>: - __repr__ method has been called.
INFO <Cargo>: - __repr__ method has been called.
INFO <Container>: - __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

Act Basic: Task 04

NAME

Inheritance

DIRECTORY

`t04_inheritance/`

SUBMIT

`gadgets.py`

BEFORE YOU BEGIN

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

- What is inheritance?
- What is the difference between inheritance and composition?
- What does it mean to have an 'is a' relation (in programming)?
- What are the benefits of using inheritance?
- What class do all classes in Python inherit by default?
- What is multiple inheritance?
- How to call the `__init__()` method of the immediate parent class without specifying the name of the parent class?

DESCRIPTION

In this task you will have to edit an existing script in such a way that, when imported, it generates a certain output.

Take the script s05t04_inheritance_gadgets.py from the resources, and edit the classes within it. You will have to add inheritance to the defined classes and make the methods work in the expected way. In each class you're allowed to edit the class definition line, and the contents of the methods. Don't add/remove methods or classes. Follow directions inside the script on what can be edited.

If everything is correct, the s05t04_inheritance_main.py script must produce output as shown in the CONSOLE VIEW. Don't forget to rename s05t04_inheritance_gadgets.py to gadgets.py.

EXAMPLE

s05t04_inheritance_main.py

```
from gadgets import Phone, Computer, Smartphone, IPhone


def test_instance(instance):
    print(instance.__class__)
    print(*instance.__dict__.items(), sep='\n')
    if hasattr(instance, 'make_call'):
        instance.make_call('+380 72 384 4834')
    print('-')


if __name__ == '__main__':
    phone = Phone(number='+380 50 709 3941')
    test_instance(phone)
    computer = Computer(operating_system='Windows 10',
                        cpu='Intel Core i7',
                        ram_size=16,
                        input_devices=['keyboard', 'mouse'])
    test_instance(computer)
    smartphone = Smartphone(operating_system='Android',
                            cpu='Octa-core',
                            ram_size=8,
                            number='+380 73 472 8879',
                            battery=4500)
    test_instance(smartphone)
    iphone = IPhone(cpu='Hexa-core',
                    ram_size=6,
                    number='+380 95 843 8467',
                    battery=2815)
    test_instance(iphone)
```

s05t04_inheritance_gadgets.py

```
# colors to prettify output, don't edit
clr = ['\033[38;5;208m',  # Phone
       '\033[38;5;112m',  # Computer
       '\033[38;5;87m',   # Smartphone
       '\033[38;5;160m',  # IPhone
       '\033[0m']


class Phone(object):  # you may edit within the parentheses
    def __init__(self, number):
        print(f'{clr[0]}[Phone init ({self.__class__.__name__})]{clr[4]}')
```

```
        # don't edit above
        # write your code here (only inside the `__init__` method)

    def make_call(self, number):
        print(f'{clr[0]}[Phone make call ({self.__class__.__name__})]{clr[4]}')
        # don't edit above
        # write your code here (only inside the `make_call` method)


class Computer(object):  # you may edit within the parentheses
    def __init__(self, operating_system, cpu, ram_size, input_devices):
        print(f'{clr[1]}[Computer init ({self.__class__.__name__})]{clr[4]}')
        # don't edit above
        # write your code here (only inside the `__init__` method)


class Smartphone(object):  # you may edit within the parentheses
    def __init__(self, operating_system, cpu, ram_size, number, battery):
        print(f'{clr[2]}[Smartphone init ({self.__class__.__name__})]{clr[4]}')
        # don't edit above
        # write your code here (only inside the `__init__` method)


class IPhone(object):  # you may edit within the parentheses
    def __init__(self, cpu, ram_size, number, battery):
        print(f'{clr[3]}[IPhone init ({self.__class__.__name__})]{clr[4]}')
        # don't edit above
        # write your code here (only inside the `__init__` method)
```

CONSOLE VIEW

```
>python3 s05t04_inheritance_main.py
[Phone init (Phone)]
<class 'gadgets.Phone'>
('number', '+380 50 709 3941')
[Phone make call (Phone)]
Call from +380 50 709 3941 to +380 72 384 4834.
-
[Computer init (Computer)]
<class 'gadgets.Computer'>
('operating_system', 'Windows 10')
('cpu', 'Intel Core i7')
('ram_size', 16)
('input_devices', ['keyboard', 'mouse'])
-
[Smartphone init (Smartphone)]
[Computer init (Smartphone)]
[Phone init (Smartphone)]
```

```
<class 'gadgets.Smartphone'>
('operating_system', 'Android')
('cpu', 'Octa-core')
('ram_size', 8)
('input_devices', ['touch screen'])
('number', '+380 73 472 8879')
('battery', 4500)
[Phone make call (Smartphone)]
Call from +380 73 472 8879 to +380 72 384 4834.
-
[IPhone init (IPhone)]
[Smartphone init (IPhone)]
[Computer init (IPhone)]
[Phone init (IPhone)]
<class 'gadgets.IPhone'>
('operating_system', 'iOS')
('cpu', 'Hexa-core')
('ram_size', 6)
('input_devices', ['touch screen'])
('number', '+380 95 843 8467')
('battery', 2815)
[Phone make call (IPhone)]
Call from +380 95 843 8467 to +380 72 384 4834.
-
>
```

SEE ALSO

Classes - Inheritance
Python super()

Act Basic: Task 05

NAME

Threads

DIRECTORY

`t05_threads/`

SUBMIT

`threads.py`

BEFORE YOU BEGIN

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

- What is a thread in programming?
- What is multithreading (in programming)?
- Which module must be used to work with threads in Python?
- How to create a thread? What arguments are required?
- How to start a thread?

DESCRIPTION

Create a script that reads files using threads.
The script consists of two functions:

- `start_threads()` that takes a list of dictionaries of thread information, creates and starts a thread for every item in the given list
  When creating a new thread it uses: `task_thread()` as the function to be executed by every thread, and tha values of `'name'`, `'path'`, `'delay'` of the item as the arguments to be passed to the target function
- `task_thread()` takes the thread's arguments (`name`, `path`, and `delay`), prints the contents of the file at the given path line by line, and sleeps after every line for the `delay` number of seconds

Use the `s05t05_threads_main.py` script to test your solution. It must produce output as shown in the `CONSOLE VIEW`.

EXAMPLE

```
from threads import task_thread, start_threads


if __name__ == '__main__':

    jobs = [
        {'name': 'Thread 1', 'path': 's05t05_threads_file1.txt', 'delay': 3},
        {'name': 'Thread 2', 'path': 's05t05_threads_file2.txt', 'delay': 1},
        {'name': 'Thread 3', 'path': 's05t05_threads_file1.txt', 'delay': 5},
```

```
    ]

    start_threads(jobs)
```

CONSOLE VIEW

```
>python3 s05t05_threads_main.py
[Thread 1]: file 1 line1
[Thread 2]: file 2 line1
[Thread 3]: file 1 line1
[Thread 2]: file 2 line2
[Thread 2]: file 2 line3
[Thread 1]: file 1 line2
[Thread 3]: file 1 line2
[Thread 1]: file 1 line3
[Thread 1]: file 1 line4
[Thread 3]: file 1 line3
[Thread 3]: file 1 line4
>
```

SEE ALSO

An Intro to Threading in Python
Python - Multithreaded Programming
Python - Date and Time

Act Basic: Task 06

NAME

Processes

DIRECTORY

`t06_processes/`

SUBMIT

`processes.py`

BEFORE YOU BEGIN

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

- What is a process in programming?
- What is multiprocessing (in programming)?
- Which module must be used to work with processes in Python?
- How to create a process? What arguments are required?
- How to start a process?
- What are the differences in working with threads and processes?

DESCRIPTION

Create a script that works with processes.
The script consists of two functions:

- `start_processes()` that takes a list of dictionaries with process information, creates and starts a process for every item in the given list
  When creating a new process it uses `task_process()` as the function to be executed by every process. Takes `'name'`, `'year'`, `'delay'` as the arguments to be passed to the target function
- `task_process()` takes the process's arguments (`name`, `year`, `delay`), prints the name and age of the person in the current year, and sleeps for the `delay` number of seconds

Use the `s05t06_processes_main.py` script to test your solution. It must produce output as shown in the `CONSOLE VIEW`.

EXAMPLE

```
from processes import task_process, start_processes


if __name__ == '__main__':
        jobs = [
            {'name': 'John', 'year': 1990, 'delay': 10},
            {'name': 'Chris', 'year': 1993, 'delay': 5},
            {'name': 'Matthew', 'year': 2010, 'delay': 3},
        ]

        start_processes(jobs)
```

CONSOLE VIEW

```
>python3 s05t06_processes_main.py
Matthew, 11 years old
Chris, 28 years old
John, 31 years old
>
```

SEE ALSO

Multiprocessing Vs. Threading In Python: What You Need To Know.
multiprocessing - Process-based parallelism

Act Advanced: Task 07

NAME

Dynamic

DIRECTORY

`t07_dynamic/`

SUBMIT

`witch_maker.py`

BEFORE YOU BEGIN

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

- How to create a class dynamically?
- In what real-life cases would it be useful to create classes dynamically?
- How to get the name of a function?

DESCRIPTION

Create a class `Witch` and a function `get_witch_class()` that dynamically creates and returns a child class of `Witch`.

The class `Witch` must have an `__init__()` method that takes a parameter `name` and sets the instance's `name` to that value.

The function `get_witch_class()` takes three positional arguments:

- `class_name` - the name of the class to be created
- `specialty` - the value of the `specialty` class attribute
- `skills` - a list of functions that must become the methods of the new class

The function must create a class with the name given as the `class_name` argument. The newly created class must inherit the class `Witch`. The class must have an attribute `specialty`, set to the value given as the `specialty` argument. Also, the class must have the methods given as a list in the `skills` attribute.

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 witch_maker.py, not the test script.

EXAMPLE

```
from witch_maker import get_witch_class


# function to test a created class and its instance
def test(my_class, instance):
    # get __dict__ of class without special methods and attributes
    class_dict = list(filter(lambda x: not x[0].startswith('__'),
                             my_class.__dict__.items()))
    methods = list(filter(lambda x: callable(x[1]), class_dict))
    print(f'class {my_class.__name__} ({my_class.__base__.__name__})')
    print('class __dict__:', *class_dict)
    print('instance __dict__:', instance.__dict__)
    for method_name, method in methods:
        print(f'Calling method "{method_name}": ', end='')
        getattr(instance, method_name)()
    print('---')


# functions that will be added to 'skills'
def control_tides(self):
    print(f'{self.name} is controlling the tides.')


def read_tarot(self):
    print(f'{self.name} is reading tarot cards.')


def palmistry(self):
    print(f'{self.name} is palm reading.')


def conjure_dead(self):
    print(f'{self.name} is conjuring the spirits of the dead.')


if __name__ == "__main__":
    NoSkillWitch = get_witch_class('NoSkillWitch', None, [])
    test(NoSkillWitch, NoSkillWitch('Diana'))

    SeaWitch = get_witch_class('SeaWitch', 'water magic', [control_tides])
    test(SeaWitch, SeaWitch('Calypso'))

    DivinationWitch = get_witch_class('DivinationWitch', 'fortune telling',
                                      [read_tarot, palmistry])
    test(DivinationWitch, DivinationWitch('Ada'))

    Necromancer = get_witch_class('Necromancer', 'the dead', [conjure_dead])
    test(Necromancer, Necromancer('the Witch of Endor'))
```

```
    TestWitch = get_witch_class('L', 'lambda methods',
                                [lambda self:
                                 print(f'{self.name} is in lambda.')])
    test(TestWitch, TestWitch('Ella'))
```

CONSOLE VIEW

```
>python3 s05t07_dynamic_main.py
class NoSkillWitch (Witch)
class __dict__: ('specialty', None)
instance __dict__: {'name': 'Diana'}
---
class SeaWitch (Witch)
class __dict__: ('specialty', 'water magic') ('control_tides', <function control_tides
↪  at 0x7fdbcff4c670>)
instance __dict__: {'name': 'Calypso'}
Calling method "control_tides": Calypso is controlling the tides.
---
class DivinationWitch (Witch)
class __dict__: ('specialty', 'fortune telling') ('read_tarot', <function read_tarot at
↪  0x7fdbcff4c700>) ('palmistry', <function palmistry at 0x7fdbcff4c790>)
instance __dict__: {'name': 'Ada'}
Calling method "read_tarot": Ada is reading tarot cards.
Calling method "palmistry": Ada is palm reading.
---
class Necromancer (Witch)
class __dict__: ('specialty', 'the dead') ('conjure_dead', <function conjure_dead at
↪  0x7fdbcff4c820>)
instance __dict__: {'name': 'the Witch of Endor'}
Calling method "conjure_dead": the Witch of Endor is conjuring the spirits of the dead.
---
class L (Witch)
class __dict__: ('specialty', 'lambda methods') ('<lambda>', <function <lambda> at
↪  0x7fdbcff4c8b0>)
instance __dict__: {'name': 'Ella'}
Calling method "<lambda>": Ella is in lambda.
---
>
```

PYTHON INTERPRETER

```
>python3
>>> from witch_maker import Witch, get_witch_class
>>> Witch.__dict__
mappingproxy({'__module__': 'witch_maker', '__init__': <function Witch.__init__ at
↪  0x7f6a30f9d9d0>, '__dict__': <attribute '__dict__' of 'Witch' objects>,
↪  '__weakref__': <attribute '__weakref__' of 'Witch' objects>, '__doc__': None})
>>> witch = Witch('test')
>>> witch.__dict__
{'name': 'test'}
>>> def detect_humans(self):
...     print(f'{self.name} has detected human scent nearby')
...
>>> BabaYaga = get_witch_class('BabaYaga', 'child-eating', [detect_humans])
>>> BabaYaga
<class 'witch_maker.BabaYaga'>
>>> yaga = BabaYaga('Yaga')
>>> yaga
<witch_maker.BabaYaga object at 0x7f6a31149730>
>>> yaga.detect_humans()
Yaga has detected human scent nearby
>>>
>>> # test with duplicate functions
>>> Test = get_witch_class('Test', 'test specialty',
...     [lambda self: 'first', lambda self: 'second'])
>>> Test.__dict__
mappingproxy({'specialty': 'test specialty', '<lambda>': <function <lambda> at
↪  0x7f6a30f9db80>, '__module__': 'witch_maker', '__doc__': None})
>>> test = Test('test')
>>> getattr(test, '<lambda>')()
'second'
>>>
```

SEE ALSO

Create Classes Dynamically in Python

Act Advanced: Task 08

NAME

New

DIRECTORY

`t08_new/`

SUBMIT

`knight.py`

BEFORE YOU BEGIN

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

- What is the purpose of the method `__new__()`?
- What does `__new__()` return?
- How to count the instances of a class?

DESCRIPTION

In your script, you will need a logger, a logger decorator, and a class `Knight`.

Use the class `Knight` that you have created for the Task 01 Knight. Add a magic method `__new__()` to the class. Use this method to forbid instantiating the class if there are already four instances made, or if the constructor receives a named argument `name` with the value `'Arthur'`. On attempts to do so, log a corresponding error message (see in CONSOLE VIEW). Pay attention that your logger must log messages to the stdout and in the format visible in the CONSOLE VIEW.

Add a class attribute `instances` to store all created instances.

Create a decorator `log()`. Apply the decorator on each method of the `Knight` class. The decorator must log the name of the called method and the passed `**kwargs` in the format visible in the CONSOLE VIEW.

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

```
from knight import Knight


if __name__ == '__main__':
    Knight(name='Robin', title='Not-quite-so-brave-as-Sir-Lancelot',
           deed='Nearly fought the Dragon of Angnor.')
    Knight(name='Arthur', age=25)
```

```
    Knight(name='Aban', height=150)
    Knight(name='Ector', title='Responsible')
    Knight(name='Galahad')
    Knight(title='Brave', skill='archery')
    Knight(name='Tristan')
    Knight(name='Arthur')
    for i in Knight.instances:
        print(i.__dict__)
```

CONSOLE VIEW

```
>python3 s05t08_new_main.py
INFO __new__ with {'name': 'Robin', 'title': 'Not-quite-so-brave-as-Sir-Lancelot',
↪  'deed': 'Nearly fought the Dragon of Angnor.'}
INFO __init__ with {'name': 'Robin', 'title': 'Not-quite-so-brave-as-Sir-Lancelot',
↪  'deed': 'Nearly fought the Dragon of Angnor.'}
INFO __new__ with {'name': 'Arthur', 'age': 25}
ERROR Cannot create a Knight with the name Arthur. Arthur is the King.
INFO __new__ with {'name': 'Aban', 'height': 150}
INFO __init__ with {'name': 'Aban', 'height': 150}
INFO __new__ with {'name': 'Ector', 'title': 'Responsible'}
INFO __init__ with {'name': 'Ector', 'title': 'Responsible'}
INFO __new__ with {'name': 'Galahad'}
INFO __init__ with {'name': 'Galahad'}
INFO __new__ with {'title': 'Brave', 'skill': 'archery'}
ERROR Cannot create a Knight. The Round Table can only fit 4 Knights.
INFO __new__ with {'name': 'Tristan'}
ERROR Cannot create a Knight. The Round Table can only fit 4 Knights.
INFO __new__ with {'name': 'Arthur'}
ERROR Cannot create a Knight. The Round Table can only fit 4 Knights.
{'name': 'Robin', 'title': 'Not-quite-so-brave-as-Sir-Lancelot', 'deed': 'Nearly fought
↪  the Dragon of Angnor.'}
{'name': 'Aban', 'height': 150}
{'name': 'Ector', 'title': 'Responsible'}
{'name': 'Galahad'}
>
```

PYTHON INTERPRETER

```
>python3
>>> from knight import Knight
>>> Knight(a='A')
INFO __new__ with {'a': 'A'}
INFO __init__ with {'a': 'A'}
<knight.Knight object at 0x7f19c220b8e0>
>>> Knight(b='B')
```

```
INFO __new__ with {'b': 'B'}
INFO __init__ with {'b': 'B'}
<knight.Knight object at 0x7f19c20f8940>
>>> Knight(name='Arthur')
INFO __new__ with {'name': 'Arthur'}
ERROR Cannot create a Knight with the name Arthur. Arthur is the King.
>>> Knight(c='C')
INFO __new__ with {'c': 'C'}
INFO __init__ with {'c': 'C'}
<knight.Knight object at 0x7f19c20f8dc0>
>>> Knight(d='D')
INFO __new__ with {'d': 'D'}
INFO __init__ with {'d': 'D'}
<knight.Knight object at 0x7f19c20f8850>
>>> Knight(e='E')
INFO __new__ with {'e': 'E'}
ERROR Cannot create a Knight. The Round Table can only fit 4 Knights.
>>> Knight(name='Arthur')
INFO __new__ with {'name': 'Arthur'}
ERROR Cannot create a Knight. The Round Table can only fit 4 Knights.
>>> Knight.instances
[<knight.Knight object at 0x7f19c220b8e0>, <knight.Knight object at 0x7f19c20f8940>,
↪  <knight.Knight object at 0x7f19c20f8dc0>, <knight.Knight object at 0x7f19c20f8850>]
>>>
```

SEE ALSO

Constructors in Python (__init__ vs __new__)

Act Advanced: Task 09

NAME

With

DIRECTORY

`t09_with/`

SUBMIT

`redirect.py`

BEFORE YOU BEGIN

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

- What is the main purpose of using context managers?
- What context managers have you worked with so far?
- What are the different ways to redirect standard output or error stream to a file?

There are many ways to redirect standard streams, try multiple strategies.

DESCRIPTION

Create a script that contains a class called `Redirect` that can be used as a context manager. The class initializer takes a string path to a file (to which you will redirect the standard stream), and a string that specifies what stream(s) to redirect from (either `'o'` for stdout, `'e'` for stderr, or `'oe'` for both).

The class must have the two magic methods that allow the use of the `with` statement on a class. Research what two methods are necessary for that.

The `Redirect` context manager must redirect the specified stream(s) to the file given as a path when an instance is initialized. The file must be appended to, not overwritten.

The script in the EXAMPLE tests your function. If everything is correct, it should generate output as seen in the CONSOLE VIEW. Pay attention that you must only submit the file `redirect.py`, not the test script.

EXAMPLE

```
import sys
from string import ascii_uppercase
from redirect import Redirect


if __name__ == '__main__':
    path = 'test.txt'
    letter = iter(ascii_uppercase)
    print(next(letter), '[out]')
    print(next(letter), '[err]', file=sys.stderr)
    with Redirect(path, 'o'):
        print(next(letter), '[out]', 'in Redirect')
        print(next(letter), '[err]', 'in Redirect', file=sys.stderr)
    print(next(letter), '[out]')
    print(next(letter), '[err]', file=sys.stderr)
    with Redirect(path, 'e'):
        print(next(letter), '[out]', 'in Redirect')
        print(next(letter), '[err]', 'in Redirect', file=sys.stderr)
    print(next(letter), '[out]')
    print(next(letter), '[err]', file=sys.stderr)
    with Redirect(path, 'oe'):
        print(next(letter), '[out]', 'in Redirect')
        print(next(letter), '[err]', 'in Redirect', file=sys.stderr)
    print(next(letter), '[out]')
    print(next(letter), '[err]', file=sys.stderr)
```

CONSOLE VIEW

```
>cat test.txt
cat: test.txt: No such file or directory
>python3 s05t09_with_main.py
A [out]
B [err]
D [err] in Redirect
E [out]
F [err]
G [out] in Redirect
I [out]
J [err]
M [out]
N [err]
>cat test.txt
C [out] in Redirect
H [err] in Redirect
K [out] in Redirect
L [err] in Redirect
>python3 s05t09_with_main.py
```

```
A [out]
B [err]
D [err] in Redirect
E [out]
F [err]
G [out] in Redirect
I [out]
J [err]
M [out]
N [err]
>cat test.txt
C [out] in Redirect
H [err] in Redirect
K [out] in Redirect
L [err] in Redirect
C [out] in Redirect
H [err] in Redirect
K [out] in Redirect
L [err] in Redirect
>
```

SEE ALSO

with statement in Python

Act Advanced: Task 10

NAME

Traffic light

DIRECTORY

`t10_traffic_light/`

SUBMIT

`traffic.py`

LEGEND

Ah, I see you have the machine that goes 'ping!'. This is my favorite.

-- Monty Python's The Meaning of Life

BEFORE YOU BEGIN

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

- What is a finite state machine?
- How does the method `__call__()` affect a class?

DESCRIPTION

Create a script that simulates a traffic light.

Create a finite state machine. Use classes:

- `TrafficLightState`
- `TrafficLightStateMachine`
- `GreenState`
- `RedState`
- `YellowState`

Switch states after 0.2 seconds. All state classes must extend the parent `TrafficLightState` class.

The states must change with the following logic:
`Green -> Yellow -> Red -> Yellow -> Green -> Yellow -> Red -> Yellow`

The state machine must work with the `TrafficLightState` class.

All actions of the state machine must be logged to the stdout using Python logging (see exact formatting in the CONSOLE VIEW).
Messages about the current state must be printed in the corresponding color:
'1;32m' (green), '[1;33m' (yellow), '[1;31m' (red), '[m' (end).

EXAMPLE

```
import sys
from traffic import TrafficLightState, GreenState, RedState, \
                    YellowState, TrafficLightStateMachine, logger


if __name__ == '__main__':
    iterations = int(sys.argv[1])
    state_machine = TrafficLightStateMachine()
    runner = state_machine()
    for i, message in enumerate(runner):
        if i >= iterations:
            break
        logger.warning(f'The traffic light is: {message}')
```

CONSOLE VIEW

```
>python3 s05t10_traffic_light_main.py 10
INFO GreenState created
INFO RedState created
INFO YellowState created
INFO TrafficLight state machine created
WARNING The traffic light is: Green
WARNING The traffic light is: Yellow
WARNING The traffic light is: Red
WARNING The traffic light is: Yellow
WARNING The traffic light is: Green
WARNING The traffic light is: Yellow
WARNING The traffic light is: Red
WARNING The traffic light is: Yellow
WARNING The traffic light is: Green
WARNING The traffic light is: Yellow
>python3 s05t10_traffic_light_main.py 5
INFO GreenState created
INFO RedState created
INFO YellowState created
INFO TrafficLight state machine created
WARNING The traffic light is: Green
WARNING The traffic light is: Yellow
WARNING The traffic light is: Red
WARNING The traffic light is: Yellow
WARNING The traffic light is: Green
>
```

SEE ALSO

Building Finite State Machines with Python Coroutines

Act Advanced: Task 11

NAME

Bounce

DIRECTORY

`t11_bounce/`

SUBMIT

`bounce.py`

BEFORE YOU BEGIN

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

- What is a queue in programming?
- What is the functionality of `multiprocessing.Queue`?
- Why is checking the size of a `multiprocessing.Queue` sometimes not reliable?
- How to limit the size a `multiprocessing.Queue`?
- What happens if you call `get(False)` on an empty queue? And what does `False` here stand for?

DESCRIPTION

In this task, you will complete the missing parts of an already existing script. Altogether, the logic of this program is the following: create multiple processes that will, in parallel, work with the same queues of objects.

There will be two queues: `task_queue` and `done_queue`. The `task_queue` consists of different 'balls' that the processes will take, 'bounce' once, and either put back, or put it to the `done_queue`. Then take a ball again, and so on. Every ball has a certain number of times to be bounced. If the counter drops to zero, the ball should be placed into the `done_queue`.

As a simple analogy, imagine a group of kids in the gym. There is a box full of balls, and each kid every once in a while takes a ball from the box, plays with it for a bit, and then puts it back. If the ball is worn out, kids will put it into the trashcan instead of the box.

Take `s05t11_bounce_bounce.py` from the `resources` and edit it in such a way that, when imported, it generates a certain output. Follow the editing instructions inside the script's comments. Don't add/remove functions or classes.

The two functions that you will need to complete within your solution are:

- `run_processes()` that
  - takes a list of `Ball` instances, and a list of arguments to pass to the target function of the processes

  - creates a `multiprocessing.Queue` called `task_queue` (already done for you)
  - creates a `multiprocessing.Queue` called `done_queue` (almost done for you, but you have to limit the maximum size of this queue)
    Think about what maximum size can this queue possibly have.
  - creates and starts processes with `bounce()` as the target function. The target function's arguments must be: name and delay from the given list of arguments, and the two created queues
  - puts all the given balls into the `task_queue`
  - joins all created processes (done for you)
- `bounce()` that (inside the loop)
  - sleeps for the given delay
  - stops if the `done_queue` is full
  - gets a ball from the `task_queue` (with a timeout of one second)
  - updates the ball's `n_bounces`
  - puts the ball into either `task_queue` or `done_queue`
  - prints a message about what happened (see format in the CONSOLE VIEW)

If everything is correct, the s05t11_bounce_main.py script must produce output as shown in the CONSOLE VIEW. Don't forget to rename s05t11_bounce_bounce.py to bounce.py.

EXAMPLE

s05t11_bounce_main.py

```
from bounce import run_processes, Ball


if __name__ == '__main__':
    balls_for_queue = [Ball('Red ball', 3),
                       Ball('Blue ball', 1),
                       Ball('Yellow ball', 2),
                       Ball('Green ball', 3)]
    process_args = [('Bob', 3), ('Lexa', 1), ('Mike', 5)]
    run_processes(balls_for_queue, process_args)
    print('---')
    # run with only one process
    run_processes(balls_for_queue, [('Bob', 1)])
    print('---')
    # run with only one ball
    run_processes([Ball('Red ball', 5)], process_args)
```

s05t11_bounce_bounce.py

```
import time
from multiprocessing import Process, Queue
from queue import Empty


# leave class Ball as is, don't edit it
class Ball:
    def __init__(self, name, n_bounces):
        self.name = name
        self.n_bounces = n_bounces

    def __str__(self):
        return f'{self.name} ({self.n_bounces})'


# edit the function as described in comments
def bounce(name, delay, task_queue, done_queue):
    while True:
        time.sleep(delay)

        # write you code here
        # required actions:
        # - stop if done_queue is full

        try:
            pass  # replace with your code
            # required actions:
            # - get a ball from task_queue (wait for it max 1 second)
        except Empty:
            print(f'queue.Empty exception.')
        else:
            pass  # replace with your code
            # required actions:
            # - update the ball's counter
            # - put ball either back to the task_queue, or to the done_queue
            print('replace me')  # edit contents of the message


# edit the contents of this function in the allowed section
def run_processes(balls, args):
    processes = []
    task_queue = Queue()
    done_queue = Queue()  # edit to make done_queue limited to the correct size

    # write your code here
    # required actions:
    # - create and start processes for each item of args
    # - put all the balls into the task_queue
```

```
    for p in processes:
        p.join()
```

CONSOLE VIEW

```
>python3 s05t11_bounce_main.py
Lexa bounces the Red ball (2).
Lexa bounces the Blue ball (0).
Bob bounces the Yellow ball (1).
Lexa bounces the Green ball (2).
Lexa bounces the Red ball (1).
Mike bounces the Yellow ball (0).
Lexa bounces the Green ball (1).
Bob bounces the Red ball (0).
Lexa bounces the Green ball (0).
[Lexa] done_queue is full. Process will stop.
[Bob] done_queue is full. Process will stop.
[Mike] done_queue is full. Process will stop.
---
Bob bounces the Red ball (2).
Bob bounces the Blue ball (0).
Bob bounces the Yellow ball (1).
Bob bounces the Green ball (2).
Bob bounces the Red ball (1).
Bob bounces the Yellow ball (0).
Bob bounces the Green ball (1).
Bob bounces the Red ball (0).
Bob bounces the Green ball (0).
[Bob] done_queue is full. Process will stop.
---
Lexa bounces the Red ball (4).
Lexa bounces the Red ball (3).
Bob bounces the Red ball (2).
Lexa bounces the Red ball (1).
Lexa bounces the Red ball (0).
[Mike] done_queue is full. Process will stop.
[Lexa] done_queue is full. Process will stop.
[Bob] done_queue is full. Process will stop.
>
```

SEE ALSO

The Idea of a Queue
Pipes and Queues

Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences. During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- a text post, as a summary of your reflection
- charts, infographics or other ways to visualize your information
- a video, either of your work, or a reflection video
- an audio podcast. Record a story about your experience
- a photo report with a small post

Helpful tools:

- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends
- YouTube - upload an exciting video
- GitHub - share and describe your solution
- Telegraph - create a post that you can easily share on Telegram
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.