(/)

Curriculum
Short Specializations
Average: **97.3%**

0x01. Caching

Back-end

By: Guillaume, CTO at Holberton School

Weight: 1

Project over - took place from Jan 2, 2024 6:00 AM to Jan 4, 2024 6:00 AM

An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 89.0/89 mandatory & 18.0/18 optional
- **Altogether: 200.0%**
  - Mandatory: 100.0%
  - Optional: 100.0%
  - Calculation: 100.0% + (100.0% * 100.0%) == **200.0%**

Background Context

In this project, you learn different caching algorithms.

Resources

Read or watch:

- Cache replacement policies - FIFO (/rltoken/fjhr6EvFeF3mWwsPQXUKdQ)
- Cache replacement policies - LIFO (/rltoken/U44RQjXp8xBtsbNIyhHIyw)
- Cache replacement policies - LRU (/rltoken/gKerxvR4dnXQYkBX2ujZiQ)
- Cache replacement policies - MRU (/rltoken/Tmk4qEBZ7QTknvbpKabWfQ)
- Cache replacement policies - LFU (/rltoken/8PEJ8L34bxhL2y--BW5zGQ)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/ritoken/-gpAdRQTx1Rb-amaz9JZhQ), **without the help of Google**:

General

- What a caching system is
- What FIFO means
- What LIFO means
- What LRU means
- What MRU means
- What LFU means
- What the purpose of a caching system
- What limits a caching system have

Requirements

Python Scripts

- All your files will be interpreted/compiled on Ubuntu 18.04 LTS using `python3` (version 3.7)
- All your files should end with a new line
- The first line of all your files should be exactly `#!/usr/bin/env python3`
- A `README.md` file, at the root of the folder of the project, is mandatory
- Your code should use the `pycodestyle` style (version 2.5)
- All your files must be executable
- The length of your files will be tested using `wc`
- All your modules should have a documentation (`python3 -c 'print(__import__("my_module").__doc__)'`)
- All your classes should have a documentation (`python3 -c 'print(__import__("my_module").MyClass.__doc__)'`)
- All your functions (inside and outside a class) should have a documentation (`python3 -c 'print(__import__("my_module").my_function.__doc__)'` and `python3 -c 'print(__import__("my_module").MyClass.my_function.__doc__)'`)
- A documentation is not a simple word, it's a real sentence explaining what's the purpose of the module, class or method (the length of it will be verified)

More Info

Parent class `BaseCaching`

All your classes must inherit from `BaseCaching` defined below:

(7)

```
$ cat base_caching.py
#!/usr/bin/python3
""" BaseCaching module
"""

class BaseCaching():
    """ BaseCaching defines:
      - constants of your caching system
      - where your data are stored (in a dictionary)
    """
    MAX_ITEMS = 4

    def __init__(self):
        """ Initiliaze
        """
        self.cache_data = {}

    def print_cache(self):
        """ Print the cache
        """
        print("Current cache:")
        for key in sorted(self.cache_data.keys()):
            print("{}: {}".format(key, self.cache_data.get(key)))

    def put(self, key, item):
        """ Add an item in the cache
        """
        raise NotImplementedError("put must be implemented in your cache class")

    def get(self, key):
        """ Get an item by key
        """
        raise NotImplementedError("get must be implemented in your cache class")
```

Tasks

0. Basic dictionary

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Create a class `BasicCache` that inherits from `BaseCaching` and is a caching system:

- You must use `self.cache_data` - dictionary from the parent class `BaseCaching`
- This caching system doesn't have limit
- `def put(self, key, item):`
    - Must assign to the dictionary `self.cache_data` the `item` value for the key `key`.

- If `key` or `item` is `None`, this method should not do anything.

(/).

- `def get(self, key):`
  - Must return the value in `self.cache_data` linked to `key`.
  - If `key` is `None` or if the `key` doesn't exist in `self.cache_data`, return `None`.

```
guillaume@ubuntu:~/0x01$ cat 0-main.py
#!/usr/bin/python3
""" 0-main """
BasicCache = __import__('0-basic_cache').BasicCache

my_cache = BasicCache()
my_cache.print_cache()
my_cache.put("A", "Hello")
my_cache.put("B", "World")
my_cache.put("C", "Holberton")
my_cache.print_cache()
print(my_cache.get("A"))
print(my_cache.get("B"))
print(my_cache.get("C"))
print(my_cache.get("D"))
my_cache.print_cache()
my_cache.put("D", "School")
my_cache.put("E", "Battery")
my_cache.put("A", "Street")
my_cache.print_cache()
print(my_cache.get("A"))

guillaume@ubuntu:~/0x01$ ./0-main.py
Current cache:
Current cache:
A: Hello
B: World
C: Holberton
Hello
World
Holberton
None
Current cache:
A: Hello
B: World
C: Holberton
Current cache:
A: Street
B: World
C: Holberton
D: School
E: Battery
Street
guillaume@ubuntu:~/0x01$
```

Repo:

- GitHub repository: `alx-backend`
- Directory: `0x01-caching`
- File: `0-basic_cache.py`

Done! | Help | Check your code | Get a sandbox | QA Review

1. FIFO caching

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Create a class `FIFOCache` that inherits from `BaseCaching` and is a caching system:

- You must use `self.cache_data` - dictionary from the parent class `BaseCaching`
- You can overload `def __init__(self):` but don't forget to call the parent init: `super().__init__()`
- `def put(self, key, item):`
  - Must assign to the dictionary `self.cache_data` the `item` value for the key `key`.
  - If `key` or `item` is `None`, this method should not do anything.
  - If the number of items in `self.cache_data` is higher that `BaseCaching.MAX_ITEMS`:
    - you must discard the first item put in cache (FIFO algorithm)
    - you must print `DISCARD:` with the `key` discarded and following by a new line
- `def get(self, key):`
  - Must return the value in `self.cache_data` linked to `key`.
  - If `key` is `None` or if the `key` doesn't exist in `self.cache_data`, return `None`.

(7)

```
guillaume@ubuntu:~/0x01$ cat 1-main.py
#!/usr/bin/python3
""" 1-main """
FIFOCache = __import__('1-fifo_cache').FIFOCache

my_cache = FIFOCache()
my_cache.put("A", "Hello")
my_cache.put("B", "World")
my_cache.put("C", "Holberton")
my_cache.put("D", "School")
my_cache.print_cache()
my_cache.put("E", "Battery")
my_cache.print_cache()
my_cache.put("C", "Street")
my_cache.print_cache()
my_cache.put("F", "Mission")
my_cache.print_cache()

guillaume@ubuntu:~/0x01$ ./1-main.py
Current cache:
A: Hello
B: World
C: Holberton
D: School
DISCARD: A
Current cache:
B: World
C: Holberton
D: School
E: Battery
Current cache:
B: World
C: Street
D: School
E: Battery
DISCARD: B
Current cache:
C: Street
D: School
E: Battery
F: Mission
guillaume@ubuntu:~/0x01$
```

Repo:

- GitHub repository: `alx-backend`
- Directory: `0x01-caching`
- File: `1-fifo_cache.py`

Done! | Help | Check your code | Get a sandbox | QA Review

2. LIFO Caching

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Create a class `LIFOCache` that inherits from `BaseCaching` and is a caching system:

- You must use `self.cache_data` - dictionary from the parent class `BaseCaching`
- You can overload `def __init__(self):` but don't forget to call the parent init: `super().__init__()`
- `def put(self, key, item):`
  - Must assign to the dictionary `self.cache_data` the `item` value for the key `key`.
  - If `key` or `item` is `None`, this method should not do anything.
  - If the number of items in `self.cache_data` is higher that `BaseCaching.MAX_ITEMS`:
    - you must discard the last item put in cache (LIFO algorithm)
    - you must print `DISCARD:` with the `key` discarded and following by a new line
- `def get(self, key):`
  - Must return the value in `self.cache_data` linked to `key`.
  - If `key` is `None` or if the `key` doesn't exist in `self.cache_data`, return `None`.

(7)

```
guillaume@ubuntu:~/0x01$ cat 2-main.py
#!/usr/bin/python3
""" 2-main """
LIFOCache = __import__('2-lifo_cache').LIFOCache

my_cache = LIFOCache()
my_cache.put("A", "Hello")
my_cache.put("B", "World")
my_cache.put("C", "Holberton")
my_cache.put("D", "School")
my_cache.print_cache()
my_cache.put("E", "Battery")
my_cache.print_cache()
my_cache.put("C", "Street")
my_cache.print_cache()
my_cache.put("F", "Mission")
my_cache.print_cache()
my_cache.put("G", "San Francisco")
my_cache.print_cache()

guillaume@ubuntu:~/0x01$ ./2-main.py
Current cache:
A: Hello
B: World
C: Holberton
D: School
DISCARD: D
Current cache:
A: Hello
B: World
C: Holberton
E: Battery
Current cache:
A: Hello
B: World
C: Street
E: Battery
DISCARD: C
Current cache:
A: Hello
B: World
E: Battery
F: Mission
DISCARD: F
Current cache:
A: Hello
B: World
E: Battery
G: San Francisco
guillaume@ubuntu:~/0x01$
```

Repo:

- GitHub repository: `alx-backend`
- Directory: `0x01-caching`
- File: `2-lifo_cache.py`

Done! | Help | Check your code | Get a sandbox | QA Review

3. LRU Caching

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Create a class `LRUCache` that inherits from `BaseCaching` and is a caching system:

- You must use `self.cache_data` - dictionary from the parent class `BaseCaching`
- You can overload `def __init__(self):` but don't forget to call the parent init: `super().__init__()`
- `def put(self, key, item):`
  - Must assign to the dictionary `self.cache_data` the `item` value for the key `key`.
  - If `key` or `item` is `None`, this method should not do anything.
  - If the number of items in `self.cache_data` is higher that `BaseCaching.MAX_ITEMS`:
    - you must discard the least recently used item (LRU algorithm)
    - you must print `DISCARD:` with the `key` discarded and following by a new line
- `def get(self, key):`
  - Must return the value in `self.cache_data` linked to `key`.
  - If `key` is `None` or if the `key` doesn't exist in `self.cache_data`, return `None`.

(7)

```
guillaume@ubuntu:~/0x01$ cat 3-main.py
#!/usr/bin/python3
""" 3-main """
LRUCache = __import__('3-lru_cache').LRUCache

my_cache = LRUCache()
my_cache.put("A", "Hello")
my_cache.put("B", "World")
my_cache.put("C", "Holberton")
my_cache.put("D", "School")
my_cache.print_cache()
print(my_cache.get("B"))
my_cache.put("E", "Battery")
my_cache.print_cache()
my_cache.put("C", "Street")
my_cache.print_cache()
print(my_cache.get("A"))
print(my_cache.get("B"))
print(my_cache.get("C"))
my_cache.put("F", "Mission")
my_cache.print_cache()
my_cache.put("G", "San Francisco")
my_cache.print_cache()
my_cache.put("H", "H")
my_cache.print_cache()
my_cache.put("I", "I")
my_cache.print_cache()
my_cache.put("J", "J")
my_cache.print_cache()
my_cache.put("K", "K")
my_cache.print_cache()

guillaume@ubuntu:~/0x01$ ./3-main.py
Current cache:
A: Hello
B: World
C: Holberton
D: School
World
DISCARD: A
Current cache:
B: World
C: Holberton
D: School
E: Battery
Current cache:
B: World
C: Street
D: School
E: Battery
None
World
```

```
Street
(/)
DISCARD: D
Current cache:
B: World
C: Street
E: Battery
F: Mission
DISCARD: E
Current cache:
B: World
C: Street
F: Mission
G: San Francisco
DISCARD: B
Current cache:
C: Street
F: Mission
G: San Francisco
H: H
DISCARD: C
Current cache:
F: Mission
G: San Francisco
H: H
I: I
DISCARD: F
Current cache:
G: San Francisco
H: H
I: I
J: J
DISCARD: G
Current cache:
H: H
I: I
J: J
K: K
guillaume@ubuntu:~/0x01$
```

Repo:

- GitHub repository: `alx-backend`
- Directory: `0x01-caching`
- File: `3-lru_cache.py`

Done! | Help | Check your code | Get a sandbox | QA Review

4. MRU Caching

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

(/)

Create a class `MRUCache` that inherits from `BaseCaching` and is a caching system:

- You must use `self.cache_data` - dictionary from the parent class `BaseCaching`
- You can overload `def __init__(self):` but don't forget to call the parent init: `super().__init__()`
- `def put(self, key, item):`
  - Must assign to the dictionary `self.cache_data` the `item` value for the key `key`.
  - If `key` or `item` is `None`, this method should not do anything.
  - If the number of items in `self.cache_data` is higher that `BaseCaching.MAX_ITEMS`:
    - you must discard the most recently used item (MRU algorithm)
    - you must print `DISCARD:` with the `key` discarded and following by a new line
- `def get(self, key):`
  - Must return the value in `self.cache_data` linked to `key`.
  - If `key` is `None` or if the `key` doesn't exist in `self.cache_data`, return `None`.

(7)

```
guillaume@ubuntu:~/0x01$ cat 4-main.py
#!/usr/bin/python3
""" 4-main """
MRUCache = __import__('4-mru_cache').MRUCache

my_cache = MRUCache()
my_cache.put("A", "Hello")
my_cache.put("B", "World")
my_cache.put("C", "Holberton")
my_cache.put("D", "School")
my_cache.print_cache()
print(my_cache.get("B"))
my_cache.put("E", "Battery")
my_cache.print_cache()
my_cache.put("C", "Street")
my_cache.print_cache()
print(my_cache.get("A"))
print(my_cache.get("B"))
print(my_cache.get("C"))
my_cache.put("F", "Mission")
my_cache.print_cache()
my_cache.put("G", "San Francisco")
my_cache.print_cache()
my_cache.put("H", "H")
my_cache.print_cache()
my_cache.put("I", "I")
my_cache.print_cache()
my_cache.put("J", "J")
my_cache.print_cache()
my_cache.put("K", "K")
my_cache.print_cache()

guillaume@ubuntu:~/0x01$ ./4-main.py
Current cache:
A: Hello
B: World
C: Holberton
D: School
World
DISCARD: B
Current cache:
A: Hello
C: Holberton
D: School
E: Battery
Current cache:
A: Hello
C: Street
D: School
E: Battery
Hello
None
```

```
Street
(/)ISCARD: C
Current cache:
A: Hello
D: School
E: Battery
F: Mission
DISCARD: F
Current cache:
A: Hello
D: School
E: Battery
G: San Francisco
DISCARD: G
Current cache:
A: Hello
D: School
E: Battery
H: H
DISCARD: H
Current cache:
A: Hello
D: School
E: Battery
I: I
DISCARD: I
Current cache:
A: Hello
D: School
E: Battery
J: J
DISCARD: J
Current cache:
A: Hello
D: School
E: Battery
K: K
guillaume@ubuntu:~/0x01$
```

Repo:

- GitHub repository: `alx-backend`
- Directory: `0x01-caching`
- File: `4-mru_cache.py`

Done! | Help | Check your code | Get a sandbox | QA Review

5. LFU Caching

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

(/)

Create a class `LFUCache` that inherits from `BaseCaching` and is a caching system:

- You must use `self.cache_data` - dictionary from the parent class `BaseCaching`
- You can overload `def __init__(self):` but don't forget to call the parent init: `super().__init__()`
- `def put(self, key, item):`
  - Must assign to the dictionary `self.cache_data` the `item` value for the key `key`.
  - If `key` or `item` is `None`, this method should not do anything.
  - If the number of items in `self.cache_data` is higher that `BaseCaching.MAX_ITEMS`:
    - you must discard the least frequency used item (LFU algorithm)
    - if you find more than 1 item to discard, you must use the LRU algorithm to discard only the least recently used
    - you must print `DISCARD:` with the `key` discarded and following by a new line
- `def get(self, key):`
  - Must return the value in `self.cache_data` linked to `key`.
  - If `key` is `None` or if the `key` doesn't exist in `self.cache_data`, return `None`.

(7)

```
guillaume@ubuntu:~/0x01$ cat 100-main.py
#!/usr/bin/python3
""" 100-main """
LFUCache = __import__('100-lfu_cache').LFUCache

my_cache = LFUCache()
my_cache.put("A", "Hello")
my_cache.put("B", "World")
my_cache.put("C", "Holberton")
my_cache.put("D", "School")
my_cache.print_cache()
print(my_cache.get("B"))
my_cache.put("E", "Battery")
my_cache.print_cache()
my_cache.put("C", "Street")
my_cache.print_cache()
print(my_cache.get("A"))
print(my_cache.get("B"))
print(my_cache.get("C"))
my_cache.put("F", "Mission")
my_cache.print_cache()
my_cache.put("G", "San Francisco")
my_cache.print_cache()
my_cache.put("H", "H")
my_cache.print_cache()
my_cache.put("I", "I")
my_cache.print_cache()
print(my_cache.get("I"))
print(my_cache.get("H"))
print(my_cache.get("I"))
print(my_cache.get("H"))
print(my_cache.get("I"))
print(my_cache.get("H"))
my_cache.put("J", "J")
my_cache.print_cache()
my_cache.put("K", "K")
my_cache.print_cache()
my_cache.put("L", "L")
my_cache.print_cache()
my_cache.put("M", "M")
my_cache.print_cache()

guillaume@ubuntu:~/0x01$ ./100-main.py
Current cache:
A: Hello
B: World
C: Holberton
D: School
World
DISCARD: A
Current cache:
B: World
```

```
C: Holberton
D: School
E: Battery
Current cache:
B: World
C: Street
D: School
E: Battery
None
World
Street
DISCARD: D
Current cache:
B: World
C: Street
E: Battery
F: Mission
DISCARD: E
Current cache:
B: World
C: Street
F: Mission
G: San Francisco
DISCARD: F
Current cache:
B: World
C: Street
G: San Francisco
H: H
DISCARD: G
Current cache:
B: World
C: Street
H: H
I: I
I
H
I
H
I
H
DISCARD: B
Current cache:
C: Street
H: H
I: I
J: J
DISCARD: J
Current cache:
C: Street
H: H
I: I
```

```
K: K
(/)
DISCARD: K
Current cache:
C: Street
H: H
I: I
L: L
DISCARD: L
Current cache:
C: Street
H: H
I: I
M: M
guillaume@ubuntu:~/0x01$
```

Repo:

- GitHub repository: `alx-backend`
- Directory: `0x01-caching`
- File: `100-lfu_cache.py`

Done! | Help | Check your code | Get a sandbox | QA Review