[180921 21:46 추가]
FoodZone의 멤버 함수 is_new_order()는 테스트를 읽고 설계를 이해하기에 좀 더 경험이 필요합니다.
따라서 40페이지에 추가적인 팁을 공유합니다.
그 뒤에는 답안을 공유합니다.
실력을 위해서 팁만 확인한 후 진행하셔도 좋고,
어려우시면 바로 답안을 이용하셔도 좋습니다.

[180924 04:41 추가]
실습 시간에 같이 작성하였던 food_truck.py에 오류가 있습니다.
지금의 테스트코드로 잡지 못 하여 생각하지 못 했습니다.
43페이지에 수정된 버전을 공유합니다.

알고리즘 실습

180920 - Queue 활용

오늘의 목표

- Queue를 활용한 응용 프로그램 이해
  - Printer 예제 실습
  - 과제

오늘의 목표

- Queue를 활용한 응용 프로그램 이해
  - Printer 예제 실습
  - 과제
- 추석에 어려운 과제 내면 화나니까! 쉽게 가는 중
  - 추석이 끝나는 주에도 어려운 과제를 안 낼 계획: 배운게(이론) 별거 없음

Feedback

지난 과제: Python 3 기본 문법

- 제출: 39 / 41 (95.12%)
  - 감점: 테스트케이스 실패
  - 추가 점수: 3명
    - 사유: 게임을 더 재미있게 -> 반복 게임, 딜러와 대결, BUST효과
- 질문: 86개 (1인 최대 15개)
  - 연구실 방문: 4명

Feedback

- AKQJ 테스트케이스가 필요한가?
  - AK 받으면 블랙잭이라 더 안 받을텐데 '테스트'를 위한 테스트케이스다.
- Ace가 3개 이상일 때 체크를 못 하는데?
  - 테스트케이스가 없음

Feedback

- 과제 난이도가 제정신인가?
- 코딩 컨벤션
  - 공백, 변수 네이밍 등등

Feedback

- 13 초콜릿 필승 조건!
- dict() 설명
- assertIn(a, b)
- lower()
- 강의 시간

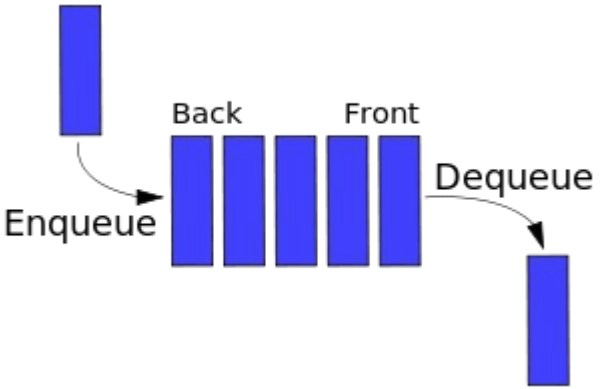
실습 및 과제: 추석 기념 윷놀이!

~~실습 및 과제: 추석 기념 윷놀이!~~

실습: Printer simulator

이론 시간에 **Queue** 활용에 대하여...

- Queue: FIFO 추상 데이터 타입
  - Stack과 함께 매우 많이 사용됨
- Queue interface
  - Enqueue
  - Dequeue



Printer simulator

- 사무실에 프린터를 하나 구매하면...
  - 평균 얼마나 기다려야할까?

```
Average Wait 846.83 secs   9 tasks remaining.
Average Wait  42.22 secs   0 tasks remaining.
Average Wait  90.25 secs   3 tasks remaining.
Average Wait  35.19 secs   0 tasks remaining.
Average Wait 103.05 secs   2 tasks remaining.
Average Wait  33.15 secs   0 tasks remaining.
Average Wait  64.25 secs   0 tasks remaining.
Average Wait 189.91 secs   0 tasks remaining.
Average Wait 271.94 secs   3 tasks remaining.
Average Wait  78.85 secs   0 tasks remaining.
```

Task class

- task.py
- 1~21장 출력

```python
import random


class Task(object):
    def __init__(self, timestamp):
        self.timestamp = timestamp
        self.pages = random.randint(1, 21)

    def get_stamp(self):
        return self.timestamp

    def get_pages(self):
        return self.pages

    def wait_time(self, current_time):
        return current_time - self.timestamp
```

Printer class

- printer.py
  - 멤버 변수
    - 분당 출력 수
  - 멤버 함수
    - 출력 진행
    - 작업중인지 확인
    - 다음 출력물 시작

```
1   class Printer(object):
2       def __init__(self, page_per_minute):
3           self.page_rate = page_per_minute
4           self.current_task = None
5           self.time_remaining = 0
6
7       def tick(self):
8           if self.current_task is not None:
9               self.time_remaining = self.time_remaining - 1
10              if self.time_remaining <= 0:
11                  self.current_task = None
12
13      def busy(self):
14          if self.current_task is not None:
15              return True
16          else:
17              return False
18
19      def start_next(self, new_task):
20          self.current_task = new_task
21          self.time_remaining = new_task.get_pages() * 60//self.page_rate
```

printer simulator

- printer_simulator.py
  - 함수들로 이루어진 소스코드
- 새로운 Job 생성
- 시뮬레이션 함수
- 메인 함수

```
28  def new_print_task():
29      num = random.randint(1, 181)
30      if num == 180:
31          return True
32      else:
33          return False
34
35
36  def main():
37      for index in range(0, 10):
38          simulation(3600, 5)
39
40
41  if __name__ == '__main__':
42      main()
```

printer simulator

```python
import random

from printer import Printer
from task import Task


def simulation(num_seconds, page_per_minute):
    lab_printer = Printer(page_per_minute)
    print_queue = list()  # Queue
    waiting_times = list()

    for current_second in range(0, num_seconds):
        if new_print_task():
            task = Task(current_second)
            print_queue.append(task)  # enqueue

        if (not lab_printer.busy()) and (len(print_queue) != 0):
            next_task = print_queue.pop(0)  # dequeue
            waiting_times.append(next_task.wait_time(current_second))
            lab_printer.start_next(next_task)

        lab_printer.tick()

    average_wait = sum(waiting_times) / len(waiting_times)
    print('Average Wait {0:6.2f} secs {1:3d} tasks remaining.'.format(average_wait, len(print_queue)))
```


백종원의
푸드트럭

들어가기에 앞서서... 우리는

- 1주차: unittest에 대하여 이해
- 2주차: unittest기반 설계를 이해하고 구현 실습, 과제
- 3주차: 한단계 더 레벨 업!
    - unittest를 읽고 설계를 이해
- Input / Output 매커니즘을 이해해야 함
    - 클래스, 함수에서 어떤 작업을 하던 I/O가 정확하면 됨

푸드트럭, 흑자날 수 있을까?

- 푸드트럭
  시뮬레이터 제작
- 주문
- 푸드트럭
- 푸드존
- 시뮬레이터

```
----# Food zone 결과(영업 시간: 1440.0, 손님 확률: 대략 300초당 1명) #----
- Food price: 2,000, making time: 60
Average wait:     159.98 secs
Order remain:            0
Total income: 103,408,000 Won
Setting cost: 40,000,000 Won
Recipe cost: 31,022,400 Won
Staff cost: 12,024,000 Won
Profit: 20,361,600 Won
```

푸드존 시뮬레이터 설계

- 주문 클래스
  - 주문 개수
- 푸드 트럭 클래스
  - 가격
  - 요리 시간
- 푸드 존 클래스
  - 푸드 트럭
  - 손님 큐

Order 클래스

- order_test.py
- test code를 통해 설계 이해

```python
import unittest

from order import Order

class OrderTest(unittest.TestCase):
    def setUp(self):
        self.timestamp = 48
        self.var_order = Order(self.timestamp)

    def tearDown(self):
        del self.timestamp
        del self.var_order

    def test___init__(self):
        self.assertEqual(self.timestamp, self.var_order.order_time)
        self.assertGreaterEqual(self.var_order.qty, 1)
        self.assertLessEqual(self.var_order.qty, 5)

    def test_get_order_time(self):
        self.assertEqual(self.timestamp, self.var_order.get_order_time())

    def test_get_qty(self):
        self.assertGreaterEqual(self.var_order.get_qty(), 1)
        self.assertLessEqual(self.var_order.get_qty(), 5)

    def test_wait_time(self):
        current_time = 60
        self.assertEqual(12, self.var_order.wait_time(current_time))

```

FoodTruck 클래스

- food_truck_test.py
- 무엇을 테스트하는가?

```python
import unittest

from order import Order
from food_truck import FoodTruck


class FoodTruckTest(unittest.TestCase):
    def test___init__(self):
        price = 5000
        making_time = 60
        var_food_truck = FoodTruck(price, making_time)
        self.assertEqual(price, var_food_truck.price)
        self.assertEqual(making_time, var_food_truck.making_time)
        self.assertEqual(None, var_food_truck.current_order)
        self.assertEqual(0, var_food_truck.time_remaining)

    def test_tick(self):
        price = 5000
        making_time = 60
        var_truck = FoodTruck(price, making_time)
        var_truck.current_order = Order(10)
        var_truck.time_remaining = 10
        for index in range(9, -1, -1):
            var_truck.tick()
            self.assertEqual(index, var_truck.time_remaining)
        self.assertEqual(None, var_truck.current_order)
```

FoodTruck 클래스

- food_truck_test.py
- 무엇을 테스트하는가?
- 각 함수를 모두 테스트해야 함

```
28      def test_is_busy(self):
29          price = 4000
30          making_time = 60
31          var_truck = FoodTruck(price, making_time)
32          var_truck.current_order = Order(10)
33          var_truck.time_remaining = 10
34          self.assertEqual(True, var_truck.is_busy())
35          var_truck.current_order = None
36          var_truck.time_remaining = 0
37          self.assertEqual(False, var_truck.is_busy())
38
39      def test_start_next_food(self):
40          price = 3000
41          making_time = 60
42          var_truck = FoodTruck(price, making_time)
43          next_order = Order(10)
44          var_truck.start_next_food(next_order)
45          self.assertEqual(next_order, var_truck.current_order)
46          self.assertEqual(making_time*next_order.get_qty(), var_truck.time_remaining)
47
48
49  if __name__ == '__main__':
50      unittest.main(verbosity=2)
```

FoodZone 클래스

- food_zone_test.py
- 생성자 테스트를 통해 알 수 있는 것

```python
import unittest

from food_truck import FoodTruck
from food_zone import FoodZone


class FoodZoneTest(unittest.TestCase):
    def test___init__(self):
        order_probability = 300
        var_food_zone = FoodZone(order_probability)
        self.assertEqual(set(), var_food_zone.food_trucks)
        self.assertEqual(list(), var_food_zone.order_queue)
        self.assertEqual(list(), var_food_zone.waiting_times)
        self.assertEqual(list(), var_food_zone.income)
        self.assertEqual(order_probability, var_food_zone.order_probability)
```

FoodZone 클래스

- food_zone_test .py
- 각 함수 테스트를 통해 알 수 있는 것?
- 이름도 중요!

```
17      def test_add_truck(self):
18          order_probability = 300
19          var_food_zone = FoodZone(order_probability)
20          price = 3000
21          making_time = 60
22          var_truck = FoodTruck(price, making_time)
23          var_food_zone.add_truck(var_truck)
24          self.assertIn(var_truck, var_food_zone.food_trucks)
25
26      def test_is_new_order(self):
27          order_probability = 300
28          var_food_zone = FoodZone(order_probability)
29          self.assertEqual(True, var_food_zone.is_new_order(order_probability))
30          self.assertEqual(False, var_food_zone.is_new_order(order_probability - 1))
31
32
33  if __name__ == '__main__':
34      unittest.main(verbosity=2)
```

돌려보자! 시뮬레이터

- food_main.py
- 코드를 이해해야 함!

```
42  def main():
43      order_probability = 300
44      open_seconds = 60*60*6*20*12
45      food_list = [(2000, 60)]
46      food_zone1 = FoodZone(order_probability)
47
48      for price, making_time in food_list:
49          food_zone1.add_truck(FoodTruck(price, making_time))
50
51      start_simulation(food_zone1, open_seconds)
52      print('----# Food zone 결과(영업 시간: {0:.1f}, 손님 확률: 대략 {1}초당 1명) #----'.format(
53              open_seconds/(60*60), order_probability))
54      print_food_zone_result(food_zone1, open_seconds)
55
56
57  if __name__ == '__main__':
58      main()
```

시뮬레이터 출력 함수

```
1   from order import Order
2   from food_truck import FoodTruck
3   from food_zone import FoodZone
4
5
6   def start_simulation(zone, test_second):...
21
22
23  def print_food_zone_result(zone, open_seconds):
24      average_wait = sum(zone.waiting_times) / len(zone.waiting_times)
25      remain_order = len(zone.order_queue)
26      total_income = sum(zone.income)
27      setting_cost = len(zone.food_trucks) * 40000000
28      recipe_cost = int(sum(zone.income)*0.3)
29      staff_cost = (open_seconds//(60*60)) * 8350 * len(zone.food_trucks) * 1
30      profit = total_income - setting_cost - recipe_cost - staff_cost
31      for truck in zone.food_trucks:
32          print('- Food price: {0:,d}, making time: {1:d}'.format(truck.price, truck.making_time))
33      print('Average wait: {0:10.2f} secs'.format(average_wait))
34      print('Order remain: {0:10d}'.format(remain_order))
35      print('Total income: {0:10,d} Won'.format(total_income))
36      print('Setting cost: {0:10,d} Won'.format(setting_cost))
37      print('Recipe cost: {0:10,d} Won'.format(recipe_cost))
38      print('Staff cost: {0:10,d} Won'.format(staff_cost))
39      print('Profit: {0:10,d} Won'.format(profit))
```

과제)

1. order.py 코드 완성
2. food_truck.py 코드 완성
3. food_zone.py 코드 완성
4. food_main.py 내부 start_simulation(zone, test_second) 완성
5. 문제 해결법 찾기(선택)
    a. 6개월만에 흑자로 돌아서는 방법은?
    b. 재료비 비중이 0.3->0.35가 되면 가격을 얼마나 올려야 비슷해지는가?
    c. 등등…

기타 유용한 정보

실습 숙제 제출

- 숙제 제출 기한: 2018. 09. 26. 23:59:59
  - 실습 전 날
- 파일 제목: AL_학번_이름_03.zip
  - 파일 제목 다를 시 채점 안 합니다.
  - .egg 안 됨!

실습 숙제 제출할 것

- 2가지 파일을 제출
- AL_학번_이름_숙제번호.zip
  - 소스코드
    - Pycharm을 사용했을 경우 Project 디렉터리에 .idea, venv 같은 디렉터리는 제외
    - Jupyter + IPython을 사용했을 경우 ‘File - Download as’ 에서 .py 다운로드 가능
  - AL_학번_이름_숙제번호.pdf
    - 보고서는 무조건 .pdf
    - .hwp, .doc 등 채점 안 함

실습 보고서에 들어가야할 것

- 목표(할 일)
- 과제를 해결하는 방법
  - 알아야 할 것
- 과제를 해결한 방법
  - 주요 소스코드: 굳이 소스코드 전체를 붙일 필요는 없음
- 결과화면
  - 결과화면 설명(해석), 테스트코드 통과
- 보고서는 기본적으로 '내가 숙제를 했음'을 보이는 것
  - 지나치게 대충 작성하면 의심하게 됨

출석부 및 실습 점수가 궁금하다면?

- 출석부 및 실습 채점표
  - 수업 시작 후 30분까지 지각, 이후 결석
  - 실습 딜레이 1일당: -2점
    - 딜레이 2일까지: -2
    - 이후 -1씩 추가

질문이 생기면?

- 이름: 문현수
- 전공: 통신및보안
- 과정: 석박사통합과정 8학기
- 연구실: 데이터네트워크연구실(공5633)
- 메일: munhyunsu@cs-cnu.org
- 알고리즘은 함께 해결해가는 과목이므로 과감하게 연락
- 이메일로 처리가 안 되는 급한일: 문자/전화 등

추석

용돈

짜증나게 하는 친척(동생)

언제 취직하니?

추가 자료

FoodZone.is_new_order() 팁

- 이 메소드는 기본적으로 new_print_task() 와 같은 기능을 함
  - 즉, 난수를 생성하고 기대 목적값 (new_print_task()에서는 181)과 같으면 True, 아니면 False를 반환
  - 이것을 통해 new_print_task()는 1/181 확률로 True를 반환
  - new_print_task에서는 1/self.order_probability 확률로 True를 반환
- 문제는 테스트코드에서 인자를 받아들이고 있다는 것
  - 왜일까?
  - 확률에 따라서 True/False가 반환되기 때문에 확률을 조정할 수 있는 장치가 필요함
  - 즉, 인수가 전달되지 않으면 난수를 생성하여 비교-반환
  - 인수가 전달되면 난수 생성대신 전달된 인수를 비교-반환
- 결론
  - 인수가 전달되지 않으면 난수를 생성하여 비교-반환
  - 인수가 전달되면 인수를 이용하여 비교-반환
- 다음 장에는 답안이 있으므로 실력을 위해선 혼자 1시간 이상 고민을 해볼 것

FoodZone.is_new_order() 답안

```
15      def is_new_order(self, picked_num=0):
16          if picked_num == 0:
17              picked_num = random.randint(1, self.order_probability)
18          if picked_num == self.order_probability:
19              return True
20          else:
21              return False
```

실습시간에 했던 food_truck.tick() 수정본

- is not이 이해되지 않으면 !=로 변경하여 진행

```
 8      def tick(self):
 9          if self.current_order is not None:
10              self.time_remaining = self.time_remaining - 1
11              if self.time_remaining <= 0:
12                  self.current_order = None
```