Using Object-Oriented Programming

Fighting Fantasy Battles - Characters

Starter – dice-rolling function

- Write a function that simulates the rolling of *n* dice and adding the score

```
def dice_sum(num_dice):


    return total
```

- (Extension) can you adapt your program so the user can also (optionally) input the number of sides for each dice (default = 6).

- Output

```
>>> dice_sum(1)
6
>>> dice_sum(6)
19
>>> dice_sum(100)
335
```

Dice-rolling function

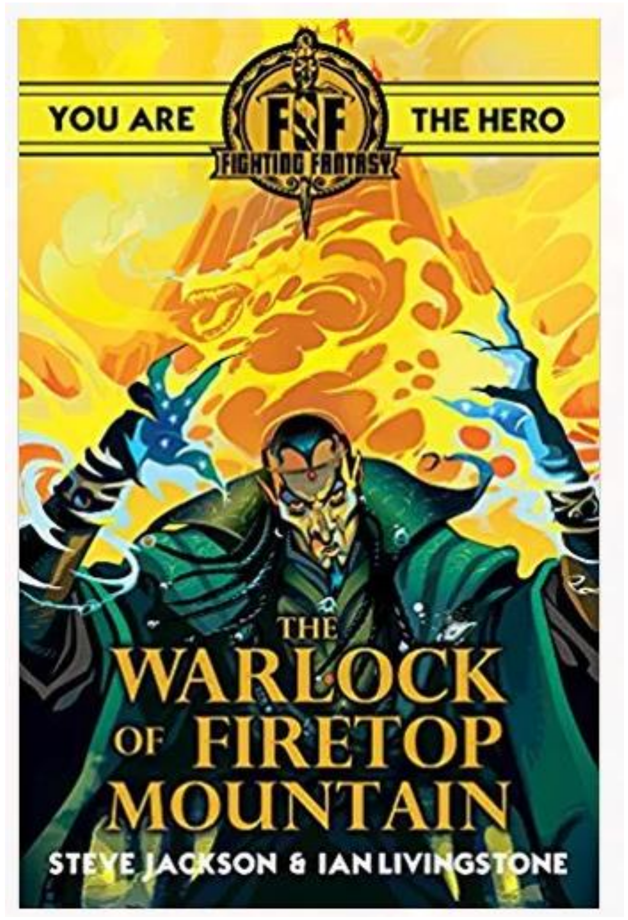
```
import random

def dice_sum(num_dice: int = 1, num_sides: int =6):
    """returns the sum of num_dice dice, each with num_sides sides"""
    total = 0
    for _ in range(num_dice):
        dice_roll = random.randint(1, num_sides)
        total += dice_roll
    return total
```

```
def dice_sum(num_dice: int = 1, num_sides: int =6):
    """returns the sum of num_dice dice, each with num_sides sides"""
    return sum(random.randint(1, num_sides) for _ in range(num_dice))
```

Fighting Fantasy

- The fighting fantasy game books had a simple fighting system
  - You and your opponent have a SKILL score
  - Each character rolls two dice and adds it to their SKILL score
  - The loser loses two from their stamina
  - If there is a draw each character losses one from their stamina
- E.g.
  - A hero, SKILL 10, STAMINA 12 vs
  - An ogre, SKILL 9, STAMINA 8


YOU ARE
THE HERO
FIGHTING FANTASY
THE WARLOCK OF FIRETOP MOUNTAIN
STEVE JACKSON & IAN LIVINGSTONE

Breaking down the problem

- To re-create Fighting Fantasy battles in Python
  - What objects could be used to represent the game?
  - What attributes would be used to represent instances of the objects?
  - How would you control the interactions between the user and the objects?

Represent a Character

- A Character object needs to have:
    - name
    - skill
    - stamina
- Write a template for the Character object
    - write an __init__ function to create instances of Character with these attributes
    - write a __repr__ function that will return a string reprenting the character, e.g.
      `Character('Dragon', skill=10, stamina=22)`

Represent a Character

```python
class Character:
    """ A Fighting Fantasy Character Object"""

    def __init__(self, name, skill=0, stamina=0):
        self.name = name
        self.skill = skill
        self.stamina = stamina

    def __repr__(self):
        return f"Character('{self.name}', skill={self.skill}, stamina={self.stamina})"
```

Create and code roll and score

- In order to fight a character will need to create a *roll* by rolling two dice and a *score* by adding the *roll* to the characters skill.
- Add `.roll` and `.score` as attributes with initial value `None`
- Write a method that will throw two dice and set the `.roll` and `.score` attributes

```
def find_score(self):
```

Create and code roll and score

```
class Character:
    """ A Fighting Fantasy Character Object"""

    def __init__(self, name, skill=0, stamina=0):
        self.name = name
        self.skill = skill
        self.stamina = stamina
        self.roll = None
        self.score = None

    def find_score(self):
        self.roll = dice_sum(num_dice=2)
        self.score = self.roll + self.skill
```

Getting wounded

- A character can be wounded in a fight.
- Create a method `.take_hit` that will reduce a character's stamina by damage.
    - The default damage is 2 but this can be overwritten

```
>>> pc
Character('Sir Andrew', skill=7,
stamina=20)
>>> pc.take_hit()
>>> pc
Character('Sir Andrew', skill=7,
stamina=18)
>>> pc.take_hit(1)
>>> pc
Character('Sir Andrew', skill=7,
stamina=17)
```

Getting wounded

```
class Character:
    """ A Fighting Fantasy Character Object"""
    …

    def take_hit(self, damage = 2):
        self.stamina -= damage
```

Fight a round

- We're now ready to fight a round. Write a method `.fight_round`
    - The arguments will be *self* and *other*, where *other* represents another character.
    - Find the score for each character
    - The loser should be wounded
    - In the case of a draw, each character should be wounded 1 Stamina
    - Keep track of the result by returning 'won', 'lost' or 'draw' from the method

Fight a round

```python
class Character:
    """ A Fighting Fantasy Character Object"""
    ...

    def fight_round(self, other):
        self.find_score()
        other.find_score()
        if self.score > other.score:
            result = 'won'
            other.take_hit()
        elif self.score < other.score:
            result = 'lost'
            self.take_hit()
        else:
            result = 'draw'
            self.take_hit(1)
            other.take_hit(1)
        return result
```

```
>>> pc
Character('Sir Andrew', skill=7, stamina=17)
>>> orc = Character('Orc', skill=6, stamina=12)
>>> pc.fight_round(orc)
'lost'
>>> pc
Character('Sir Andrew', skill=7, stamina=15)
>>> orc
Character('Orc', skill=6, stamina=12)
```

Is a character dead?

- A character is dead if their stamina is less than zero
- Write a method to determine if a character is dead

- *Use the @property decorator to make the method a property*

Is a character dead?

```
@property
def is_dead(self):
    # character.is_dead will now return True or False
    return self.stamina <= 0

@is_dead.setter
def is_dead(self, dead: bool):
    # character can be made dead or alive by setting is_dead to True or False
    if dead:
        self.stamina = 0
    else:
        self.stamina = min(self.stamina, 1)
```

Generate a PlayerCharacter

- In Fighting Fantasy there is usually a PlayerCharacter (PC) who interacts with NonPlayerCharacters (NPC)
- The PlayerCharacter has all the attributes and methods of the Character object, but will have some additional properties.
- In OOP, we can create a PlayerCharacter sub-class of Character
  - PlayerCharacter *inherits* the attributes and methods of its super-class
  - We can write new or existing attributes and methods to *overwrite* the properties of the super-class

The PlayerCharacter subclass

- The PlayerCharacter subclass will have the additional attribute *luck*
- We will also write a new *classmethod* that can be used to generate a new character with:
  - skill = 6 + 1D6
  - stamina = 12 + 2D6
  - luck = 6 + 1D6
- If, like me you don't know what 2D6 means then do some research!

```
class PlayerCharacter(Character):
    def __init__(self, name, skill=0, stamina=0, luck=0):
        super().__init__(name, skill, stamina)
        self.luck = luck

    @classmethod
    def generate_player_character(cls, name):
        # Roll for skill stamina and luck and pass them to the
            cls constructor, returning the created instance
        …
        return cls(name, skill, stamina, luck)
```