How do I Manage My Own Code?

How do I create my OWN Package!

Let’s say I wrote some cool code

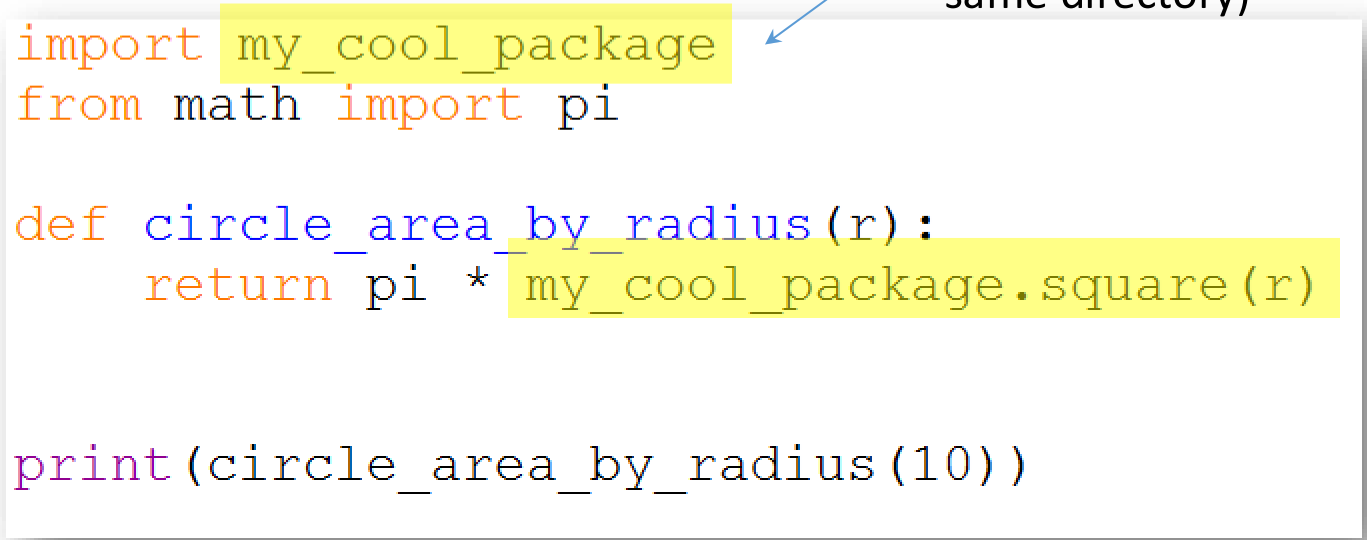
```
def square(x):
    return x * x

def singHappyBirthdayTo(name):
    print('Happy birthday To You!')
    print('Happy birthday To You!')
    print('Happy birthday to ' + name + '~')
    print('Happy birthday to You!!!')
```

- And I save it to a file called

my_cool_package.py

Then I can use it for another file

- Another file:



```
import my_cool_package
from math import pi

def circle_area_by_radius(r):
    return pi * my_cool_package.square(r)


print(circle_area_by_radius(10))
```

Same as the file name but without ".py" (provided that they are in the same directory)

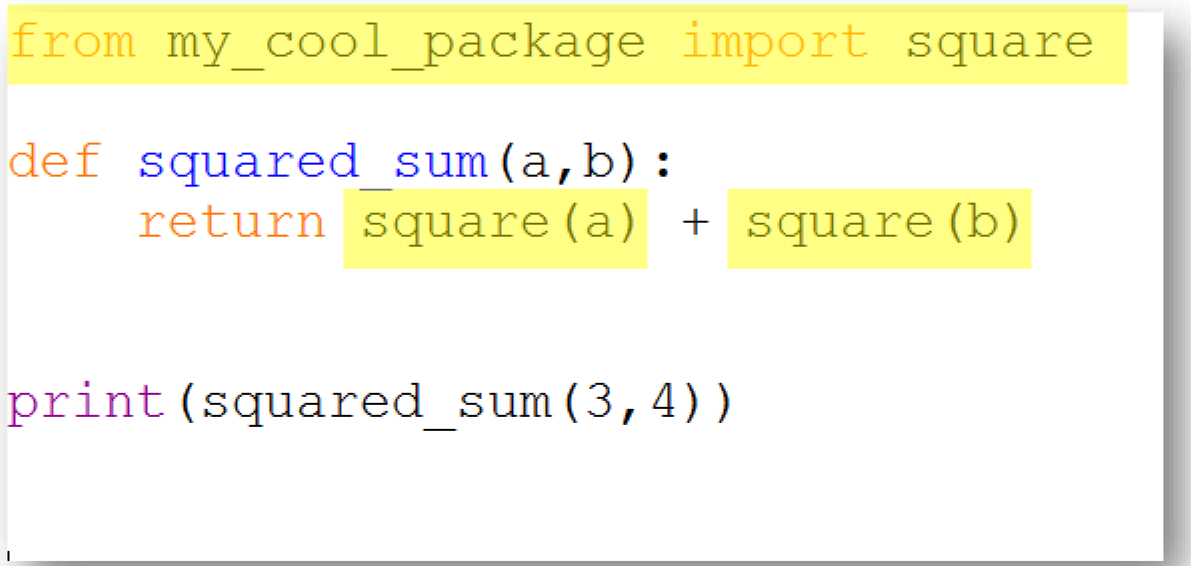
Or

- Another file:

```
import my_cool_package as mcp
from math import pi

def circle_area_by_radius(r):
    return pi * mcp.square(r)


print(circle_area_by_radius(10))
```

Or

- Another file



```
from my_cool_package import square

def squared_sum(a,b):
    return square(a) + square(b)


print(squared_sum(3,4))
```

Or

- Another file

But in general, it's not a good habit to name a variable/function so short that you cannot understand what it does

```
from my_cool_package import square as sq

def squared_sum(a,b):
    return sq(a) + sq(b)


print(squared_sum(3,4))
```

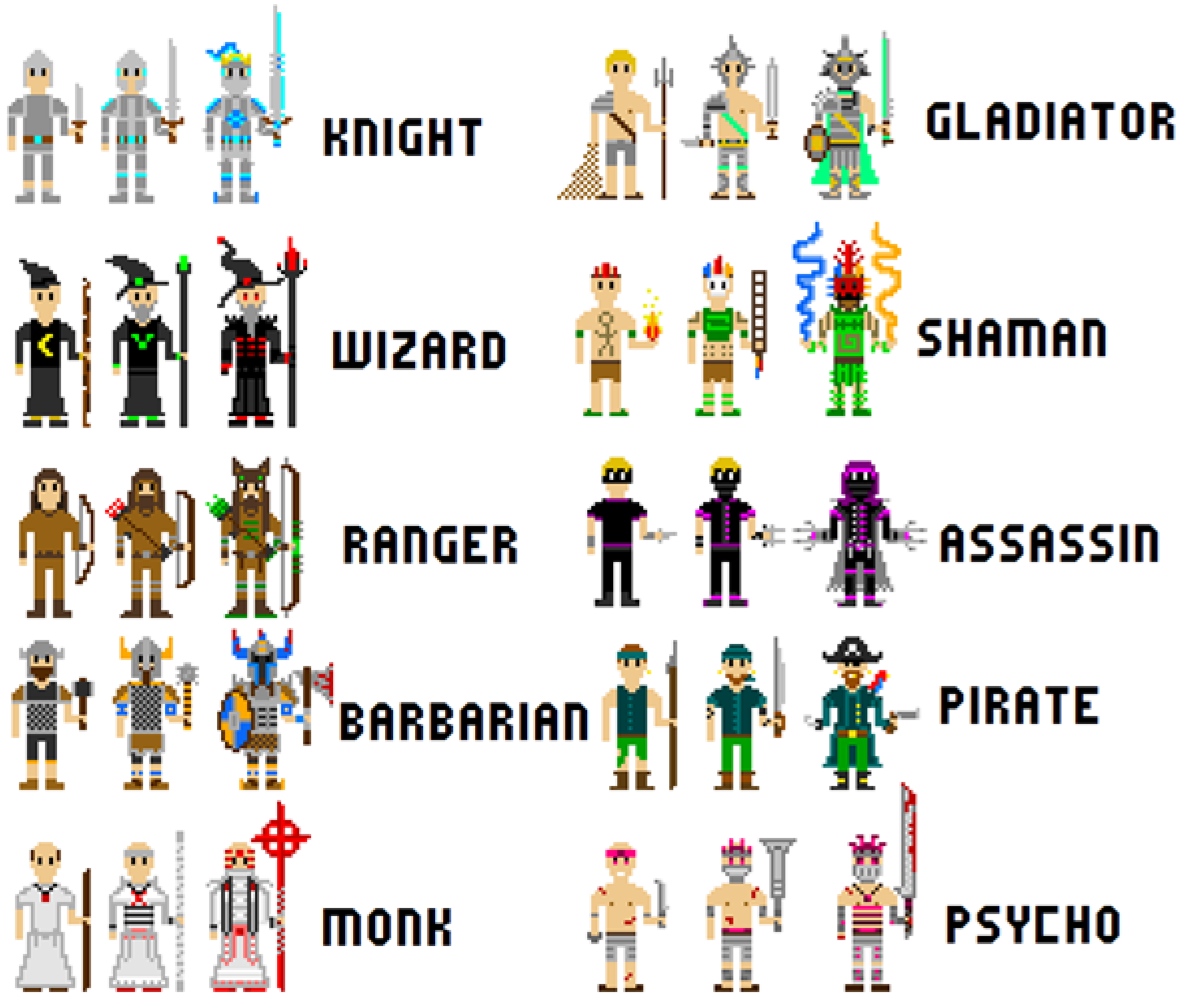
OOP Assignment: RPG!

Introduction

- We play a lot of RPG games nowadays like Final Fantasy, World of Warcrafts or DOTA. In this assignment, you are going to implement the characters for a turn-based RPG.

Type/Class of Characters

- The game is a fight between two teams of adventurers formed by different **types** of characters
  - The jargon should be "**classes**" but we do not want to be confused with the "class" in Python



Introduction

- You are given a battle system that can let two teams fight each other until one team won by destroying all the characters in the other team.

Introduction

- You are given a battle system that can let two teams fight each other until one team won by destroying all the characters in the other team.
- The good thing is that, the battle system is already implemented for you.
  - You only have to implement some new types of characters.
- You can choose to run the file **battle.py** to start the game and try it out or by
    - `gameStart(200,False)`

Before the Battle

- You are given some amount of gold (The default setting is 200 gold in the skeleton file)
- Every type of characters has a cost and you can recruit a team with any combinations of them within your budget. And each team has to use up all the gold to form.
- A random team formed by `createRandTeam()` will create a team as your enemy.
- Then you can recruit your own team by choosing available type of characters by the function `userChooseTeam()`. In the battle, you will be Team A and your enemy will be Team B.

Before the Battle

- `createRandTeam()`
- Then you can recruit your own team by choosing available type of characters by the function `userChooseTeam()`.

```
Your enemy will be:
Members:    |      Mage
Hitpoints:  |       800
Strength:   |
Max. Mana:  |        50
Current M:  |        50

Your current team:
(Currently no member in the team now)

You have 200 gold currently
Choices:
1: Fighter (cost: 100)
2: Mage (cost: 200)
Input a choice from 1 to 2:
```

Battle Starts

- Once the battle starts, each team will take turns to act.
- However, one character can perform an action for one turn.
- A character will be chosen at random within those are still alive in the team and he will perform an action targeted at a random enemy that is alive.
  - `randAlive()`
- However, the last class Necromancer has an ability to "target" your own team also!

```
THE BATTLE STARTS!!!!!

Round 1
Team A:
Members:    |     Fighter|     Fighter
Hitpoints:  |        1200|        1200
Strength:   |         100|         100
Max. Mana:  |            |
Current M:  |            |

Team B:
Members:    |        Mage
Hitpoints:  |         800
Strength:   |
Max. Mana:  |          50
Current M:  |          50

Team A member 1 Fighter acts
Hurt enemy 0 by damage 100.
Mage hurt with remaining hp 700.
```

Battle Starts

- So, when a character acts, he will hurt a targeted enemy
    - And the enemy got hurt by `gotHurt()`.
    - When the targeted enemy's hp (hitpoint) drops to zero, he will be dead and no longer acts….
    - (Unless, a necromancer revives him afterwards.)

```
THE BATTLE STARTS!!!!!

Round 1
Team A:
Members:    |     Fighter|     Fighter
Hitpoints:  |        1200|        1200
Strength:   |         100|         100
Max. Mana:  |            |
Currnet M:  |            |

Team B:
Members:    |        Mage
Hitpoints:  |         800
Strength:   |
Max. Mana:  |          50
Currnet M:  |          50

Team A member 1 Fighter acts
Hurt enemy 0 by damage 100.
Mage hurt with remaining hp 700.
```

An Example Battle

```
Round 1
Team A:
Members:   |    Fighter|       Mage|Necromancer|    Fighter
Hitpoints: |       1200|        800|        800|       1200
Strength:  |        100|           |           |        100
Max. Mana: |           |         50|         50|
Current M: |           |         50|         50|

Team B:
Members:   |   ArchMage|  Berserker
Hitpoints: |        800|       1200
Strength:  |           |        100
Max. Mana: |         50|
Current M: |         50|

Team A member 0 Fighter acts
Hurt enemy 1 by damage 100.
Berserker hurt with remaining hp 1100.
```

```
Round 2
Team A:
Members:    |     Fighter|        Mage|Necromancer|     Fighter
Hitpoints:  |        1200|         800|        800|         800
Strength:   |         100|            |           |         100
Max. Mana:  |            |          50|         50|
Current M:  |            |          50|         50|

Team B:
Members:    |    ArchMage|  Berserker
Hitpoints:  |         800|       1100
Strength:   |            |        100
Max. Mana:  |          50|
Current M:  |          30|

Team A member 0 Fighter acts
Hurt enemy 0 by damage 100.
ArchMage hurt with remaining hp 700.
(and the battle goes on….)
```

```
Team A:
Members:    |    Fighter|       Mage|Necromancer|    Fighter
Hitpoints:  |       1200|        800|        800|       1200
Strength:   |        100|           |           |        100
Max. Mana:  |           |         50|         50|
Current M:  |           |         50|         50|

Team B:
Members:    |   ArchMage|  Berserker
Hitpoints:  |        800|       1100
Strength:   |           |        100
Max. Mana:  |         50|
Current M:  |         50|

Team B member 0 ArchMage acts
Strike enemy 3 with spell
Fighter hurt with remaining hp 800.
```

Details of the Battle

- The details of the battle maybe too “annoying” and slow
- The battle information for each turn is printed by the function “`dprint()`” rather than the normal “`print()`”
    - In which, you can turn on or off by the global variable “`printActionDescription`”

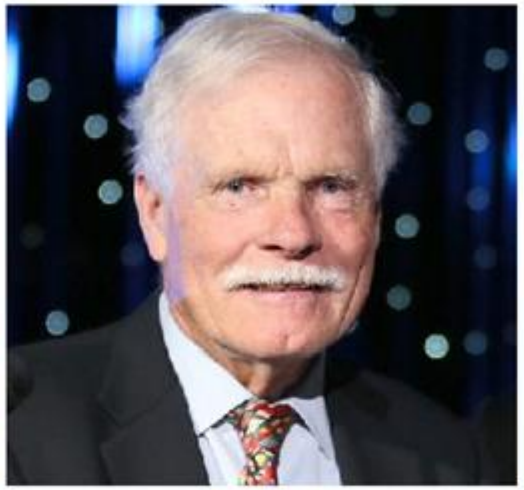
```
printActionDescription = True

def dprint(s):
    if printActionDescription:
        print(s)
```

- If “`printActionDescription`” is false, it will directly tell you which team win after the battle starts
    - Namely, skipping all the details

Character Type (Classes)

CHOOSE YOUR CLASS :

WARRIOR


PALADIN

SPECIALIST

MERCHANT

WIZARD

THIEF

Character Classes

- We will have five different character classes, namely, Fighter, Mage, Berserker, ArchMage and Necromancer.
  - Given: Fighter and Mage

Base Class: Character

- A Character is the base class of all characters
    - Initialization of the stats
    - Will do nothing
    - When he got hurt, he will either
        - Reduce his hp, or
        - Died if the damage is greater than the current hp

```
class Character(object):
    def __init__(self):
        self.name = ''
        self.maxhp = 1000
        self.hp = 1000
        self.str = 0
        self.maxmana = 0
        self.mana = 0
        self.cost = 9999999999
        self.alive = True

    def act(self,myTeam,enemy):
        return

    def gotHurt(self,damage):
        if damage >= self.hp:
            self.hp = 0
            self.alive = False
            dprint(self.name + ' died!')
        else:
            self.hp -= damage
            dprint(self.name +
                   f' hurt with remaining hp {
```

Base Class: Character

- A Character is the base class of all characters
  - Initialization of the stats
  - Will do nothing
  - When he got hurt, he will either
    - Reduce his hp, or
    - Died if the damage is greater than the current hp

```
class Character(object):
    def __init__(self):
        self.name = ''
        self.maxhp = 1000
        self.hp = 1000
        self.str = 0
        self.maxmana = 0
        self.mana = 0
        self.cost = 9999999999
        self.alive = True

    def act(self,myTeam,enemy):
        return

    def gotHurt(self,damage):
        if damage >= self.hp:
            self.hp = 0
            self.alive = False
            dprint(self.name + ' died!')
        else:
            self.hp -= damage
            dprint(self.name +
                   f' hurt with remaining hp {
```

Base Class: Character

- A Character is the base class of all characters
  - Initialization of the stats
  - Will do nothing
  - When he got hurt, he will either
    - Reduce his hp, or
    - Died if the damage is greater than the current hp

```
class Character(object):
    def __init__(self):
        self.name = ''
        self.maxhp = 1000
        self.hp = 1000
        self.str = 0
        self.maxmana = 0
        self.mana = 0
        self.cost = 9999999999
        self.alive = True

    def act(self,myTeam,enemy):
        return

    def gotHurt(self,damage):
        if damage >= self.hp:
            self.hp = 0
            self.alive = False
            dprint(self.name + ' died!')
        else:
            self.hp -= damage
            dprint(self.name +
                   f' hurt with remaining hp {
```

Base Class: Character

- A Character is the base class of all characters
  - Initialization of the stats
  - Will do nothing
  - When he got hurt, he will either
    - Reduce his hp, or
    - Died if the damage is greater than the current hp

```
class Character(object):
    def __init__(self):
        self.name = ''
        self.maxhp = 1000
        self.hp = 1000
        self.str = 0
        self.maxmana = 0
        self.mana = 0
        self.cost = 9999999999
        self.alive = True

    def act(self,myTeam,enemy):
        return

    def gotHurt(self,damage):
        if damage >= self.hp:
            self.hp = 0
            self.alive = False
            dprint(self.name + ' died!')
        else:
            self.hp -= damage
            dprint(self.name +
                   f' hurt with remaining hp {
```

Fighter Class: Subclass of Character

```
class Fighter(Character):
    def __init__(self):
        super().__init__()
        self.name = 'Fighter'
        self.maxhp = 1200
        self.hp = 1200
        self.str = 100
        self.cost = 100

    def act(self,myTeam,enemy):
        target = randAlive(enemy)
        dprint(f'Hurt enemy {target} by damage {self.str}.')
        enemy[target].gotHurt(self.str)
```

- A Fighter has a maximum hp of 1200 and strength 100.
- He has a cost of 100 gold.
- During each turn in the battle, he will select a random enemy and inflict a damage **equal** to his **strength**, i.e. 100 hp.

Fighter Class: Subclass of Character

```
class Fighter(Character):
    def __init__(self):
        super().__init__()
        self.name = 'Fighter'
        self.maxhp = 1200
        self.hp = 1200
        self.str = 100
        self.cost = 100

    def act(self,myTeam,enemy):
        target = randAlive(enemy)
        dprint(f'Hurt enemy {target} by damage {self.str}.')
        enemy[target].gotHurt(self.str)
```

- A Fighter has a maximum hp of 1200 and strength 100.
- He has a cost of 100 gold.
- During each turn in the battle, he will select a random enemy and inflict a damage **equal** to his **strength**, i.e. 100 hp.

Mage Class

- Basic Statistics
- He will cast a spell in his turn and make a damage that is equal to his intelligence.
  - However, casting a spell needs 20 mana.
  - If his remaining mana is less than that, he will take the turn to *meditate* to gain a recovery of 30 mana points instead.
  - Namely, he will not cast a spell to hurt an enemy on that turn.

```
class Mage(Character):
    def __init__(self):
        super().__init__()
        self.name = 'Mage'
        self.maxmana = 50
        self.mana = 50
        self.hp = 800
        self.cost = 200
        self.int = 400

    def act(self,myTeam,enemy):
           if self.mana < manaCost:
               self.mana += manaRecovery
               dprint(f'Mana recover to {self.mana}
           else:
               self.cast(myTeam,enemy)

    def cast(self,myTeam,enemy):
           self.mana -= manaCost
           target = randAlive(enemy)
           dprint(f'Strike enemy {target} with spel
           enemy[target].gotHurt(self.int
```

A Team

Creating a Team

- Give the amount of gold you can use, you can create a team by either one of the functions below:
  - `createRandTeam(gold)`
    - Create a random team
  - `userChooseTeam(gold)`
    - Ask the user to choose characters to “buy” into your team
- No matter which function you used, a team is a **`list`** of instances of Characters, Fighters or Mages, (or other characters you created)

Example

- Create a Random Team for the Computer

```
def createRandTeam(gold):
    team = []
    while gold > 0:
        tempChar = createChar(randint(1,nCharType))
        if gold >= tempChar.cost:
            gold -= tempChar.cost
            team.append(tempChar)
    return team
```

- Please read the code of `userChooseTeam()`.

HERE COMES
A NEW CHALLENGER!

Adding New Classes

(Berserker)

Berserker

- In the Old Norse written corpus, berserkers (or "berserks"; Old Norse: berserkir) were warriors who purportedly fought in a trance-like fury, a characteristic which later gave rise to the modern English word "berserk " or ``Berzerk `` .

Berserker Class in Our Game

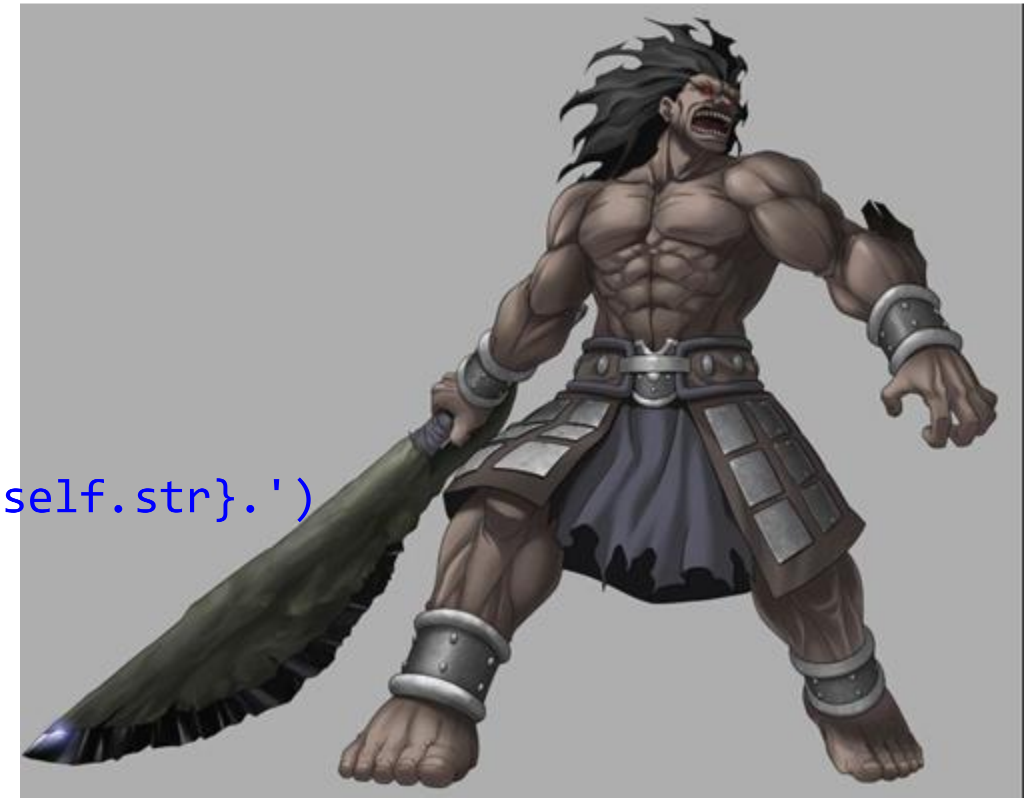
- A Berserker is a Fighter but he costs 200 gold.
- However, if his hit point is lower than or equal to half of his maximum hp, he will enter the “berserk mode” and his strength will be doubled to 200.

Berserker Class in Our Game

- Original Fighter class

```
class Fighter(Character):
    def __init__(self):
        super().__init__()
        self.name = 'Fighter'
        self.maxhp = 1200
        self.hp = 1200
        self.str = 100
        self.cost = 100

    def act(self,myTeam,enemy):
        target = randAlive(enemy)
        dprint(f'Hurt enemy {target} by damage {self.str}.')
        enemy[target].gotHurt(self.str)
```

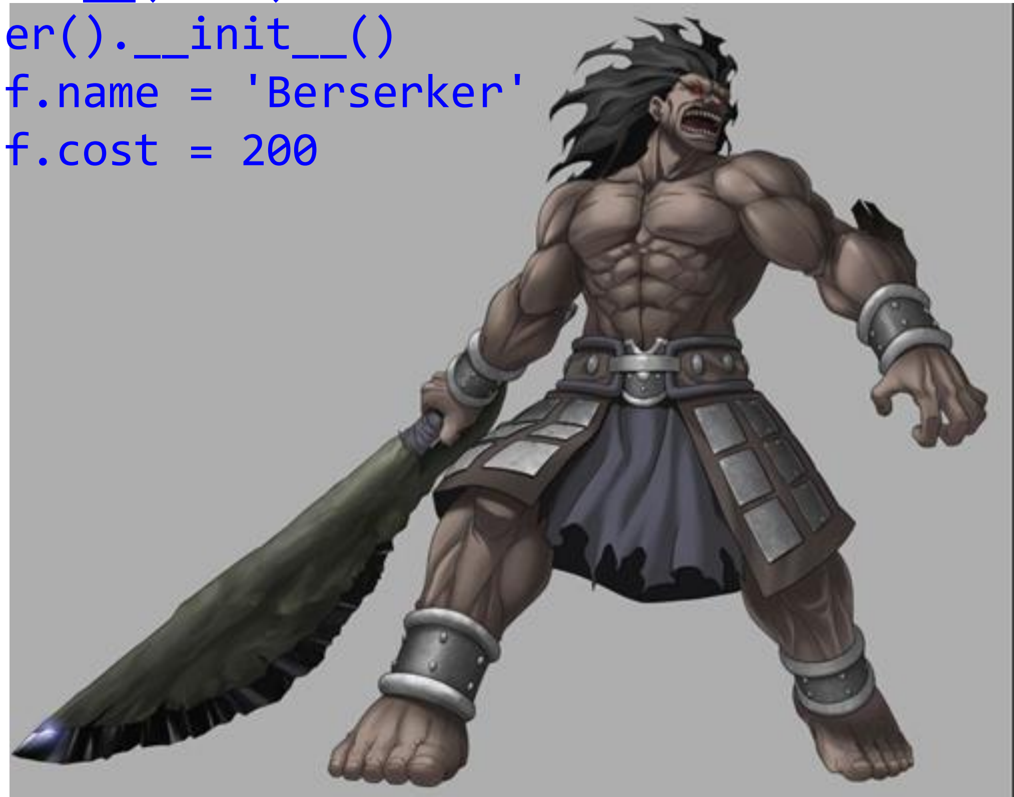
- How to modify: "*Berserker is a Fighter but he costs 200 gold*."?



Berserker Class in Our Game

- Original Fighter class
  - How to modify: " Berserker is a Fighter but he costs 200 gold. "
- First called the Fighter class constructor to set all the attributes
- Then change/overwrite the variables that is different from Fighter class

```
class Berserker(Fighter):
    def __init__(self):
        super().__init__()
        self.name = 'Berserker'
        self.cost = 200
```



Berserker Class in Our Game

- "However, if his hit point is lower than or equal to half of his maximum hp, he will enter the "berserk mode" and his strength will be doubled to 200."
- When will it happen?
  - Only when he needs to "act()"
  - Change the strength attribute if the condition is met
  - Then perform what a Fighter do

```
class Berserker(Fighter):
    def __init__(self):
        super().__init__()
        self.name = 'Berserker'
        self.cost = 200

    def act(self,myTeam,enemy):
        if self.hp <= (self.maxhp//2):
            self.str = 200
            dprint('Berserk mode! Attack double!'
        else:
            self.str = 100
        super().act(myTeam,enemy)
```

Berserker Class in Our Game

- Note the difference between the orders of calling "super()"
  - Why are they different?

```
class Berserker(Fighter):
    def __init__(self):
        super().__init__()
        self.name = 'Berserker'
        self.cost = 200

    def act(self,myTeam,enemy):
        if self.hp <= (self.maxhp//2):
            self.str = 200
            dprint('Berserk mode! Attack double!'
        else:
            self.str = 100
        super().act(myTeam,enemy)
```

Adding Berserker into the System

- The class Berserker is ready, but we need to add it to the system
- Open the `battle.py` file
- First, you need to increase the number of character type from 2 to 3

```python
from Characters import *
from random import randint
from Team import *


# You increase nCharType one at a time
nCharType = 2 # no. of types of characters you can choose
gold = 200    # Gold for you to recruit characters
minCost = 100 # A hack for userChooseTeam()

'''
Type:
1: Fighter (Given)
2: Mage (Given)
3: Berserker
4: ArchMage
5: Necromancer
'''
```

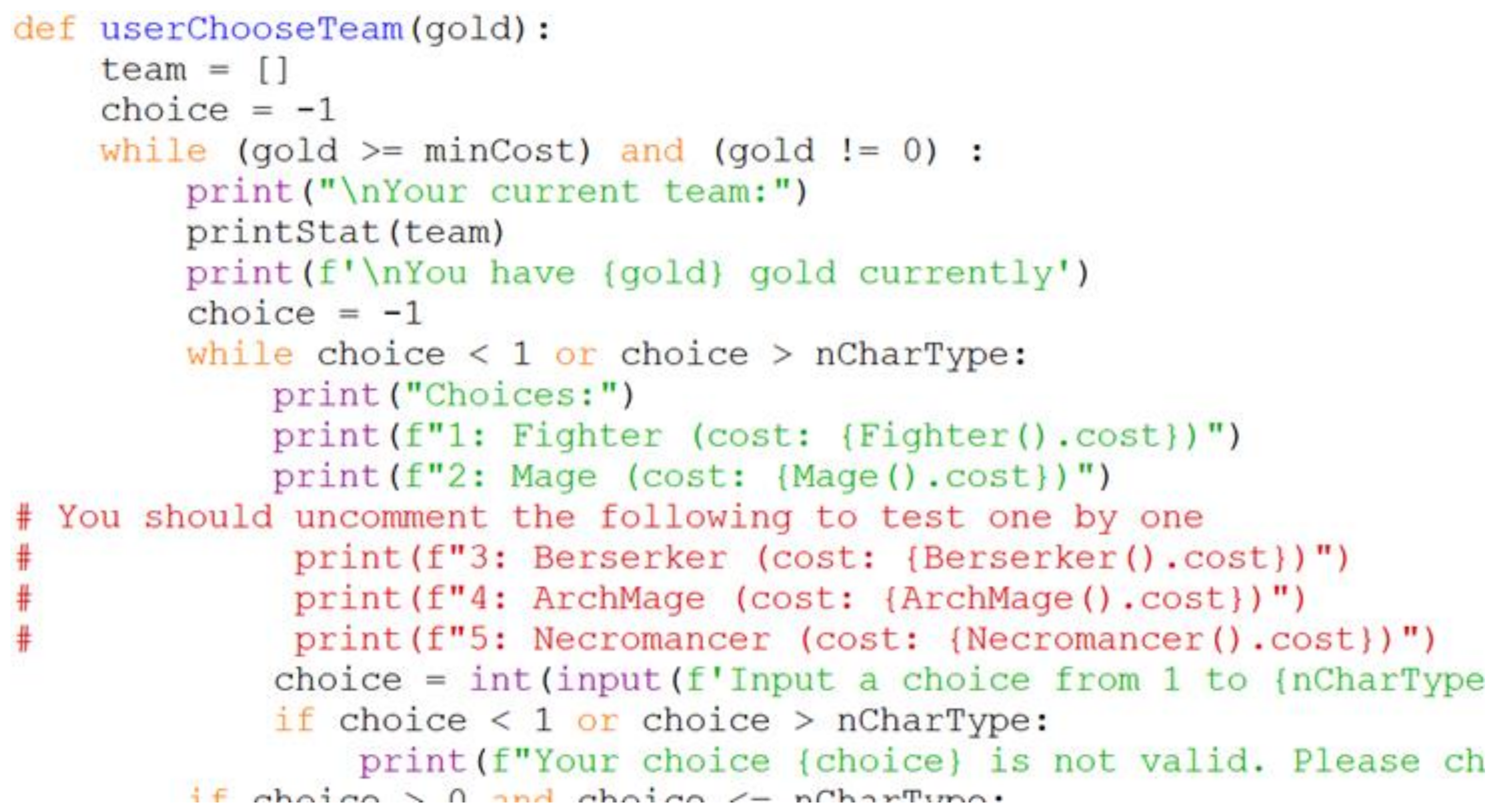
Adding Berserker into the System

- Then, allow the user to choose 3 for Berserker when they create their team
  - Uncomment these two lines
  - And these two lines ONLY!

```
def createChar(i):
    if i == 1:
        return Fighter()
    elif i == 2:
        return Mage()
# You should uncomment the following to test
'''
    elif i == 3:
        return Berserker()
    elif i == 4:
        return ArchMage()
    elif i == 5:
        return Necromancer()
'''
```

Adding Berserker into the System

- And uncomment here:



```
def userChooseTeam(gold):
    team = []
    choice = -1
    while (gold >= minCost) and (gold != 0) :
        print("\nYour current team:")
        printStat(team)
        print(f'\nYou have {gold} gold currently')
        choice = -1
        while choice < 1 or choice > nCharType:
            print("Choices:")
            print(f"1: Fighter (cost: {Fighter().cost})")
            print(f"2: Mage (cost: {Mage().cost})")
# You should uncomment the following to test one by one
#            print(f"3: Berserker (cost: {Berserker().cost})")
#            print(f"4: ArchMage (cost: {ArchMage().cost})")
#            print(f"5: Necromancer (cost: {Necromancer().cost})")
            choice = int(input(f'Input a choice from 1 to {nCharType
            if choice < 1 or choice > nCharType:
                print(f"Your choice {choice} is not valid. Please ch
        if choice > 0 and choice <= nCharType:
```

Optional

- You may want to change the default amount of gold that you can use
  - The default value is 200

```
from Characters import *
from random import randint
from Team import *


# You increase nCharType one at a time
nCharType = 2 # no. of types of characters you can choose
gold = 200    # Gold for you to recruit characters
minCost = 100 # A hack for userChooseTeam()

'''
Type:
1: Fighter (Given)
2: Mage (Given)
3: Berserker
4: ArchMage
5: Necromancer
'''
```

Finish Berserker Implementation!

If you have not done it yet

Two More Classes!

ArchMage

- If you feel like the Mage is not powerful enough

ArchMage

- An ArchMage is a Mage but he costs 600.
- However, if he is the only one alive in his own team,
  - he will cast the special spell **KABOOM** and it inflicts EVERY enemy alive with a damage of double of his intelligence, i.e. 800 hp!!!
  - However, this KABOOM spell also needs 20 mana to cast. Meaning, he has to waste a turn to meditate if his mana is lower than 20.
- Sample "act" for ArchMage

```
Team A member 1 ArchMage acts
Cast KABOOOOOOM ! (Damage 800) to every enemy!
Fighter died!
Fighter died!
```

ArchMage

- How to check “if he is the only one alive in his own team”?
- In the act() function, the two input parameters are
  - `myTeam`
  - `enemy` (Team)
- In which, they are lists of Characters
  - And in each Character, there is an attribute “alive”
- You can reference to the functions “`allAlive()`” and “`allDead()`” to understand how to check if a team is all alive or all dead

Necromancer

- A Necromancer is a Mage but he costs 400.
- However, if there are some *dead* members in his own team,
  - He can cast a spell “raise dead” with 20 mana to revive a random dead team member
  - And recover him half of his maximum hit point, instead of hurting his enemy in his turn.

```
Team A member 1 Necromancer acts
Reving member 0 with hp 500.
```

Your Tasks

- Your job is to implement the three remaining classes with inheritance.
    - Major changes all in characters.py
    - Minor changes in battle.py for adding more characters
    - No change should be done in team.py
- More details on the worksheet

Submission

- You should only copy and paste **the classes for the three new characters** into coursemology from the file characters.py.
- And there will be no testing feedback for you because the game is random.

Tips

- Design before you implement. Plan out your class inheritance diagram before you code, especially what are commons or not in each of the class relationship
- Implement one class at a time
- Make good use of the pause and `dprint()`.
- Make backup from time to time. Especially a copy of your code for each new character type you implemented
- You cannot do any hard-coding. And you have to follow the way of OOP instead using other ways to get around, e.g. checking the Character’s name to choose what to do

Extra Tasks

- Finding the Best Team
  - If you are given certain amount of gold, say 1200, what should be the best team composition?
- Create your new classes
  - Ranger? Assassin? Cleric? Bard?
  - Vampire?
    - Regenerate health when attack
  - Confused Fighter?
    - May have a %25 chance attack your own team
- Battle Formation
  - Difference rows?