Exercise Sheet 02 Solutions – Variables, Assignments, and Functions

Sebastian Höffner   Aline Vilks

Deadline: Mon, 17 Apr 2017 08:00 +0200

Exercise 1: St. Nicholas' Living Space

File: `code/nick_area.py`

```
import random


def area(base, side, height):
    return base * side + base * height / 2


area1 = area(5, 3, 2)  # should be approx. 20
print('Area 1:', area1)
area2 = area(2, 5, 1)  # should be approx. 11
print('Area 2:', area2)
area3 = area(3, 3, 3)  # should be approx. 13.5
print('Area 3:', area3)

random_side = random.randint(2, 5)
random_base = random.randint(random_side, 5)  # Constraint
random_height = random.randint(1, 3)  # Small roof

print('Random area:', area(random_base, random_side, random_height))
```

Output:

```
Area 1: 20.0
Area 2: 11.0
Area 3: 13.5
Random area: 17.5
```

Exercise 2: Castles crashed

File: code/castle_crashers.py

```python
import math


def strong_attack_damage(level, strength):
    return (5 + 1.15 * strength + 0.1 * level) // 1


def normal_attack_damage(level, strength):
    return (3 + strength + 0.1 * level) // 1


def throw_attack_damage(level, strength):
    return (10 + 1.2 * strength + 0.1 * level) // 1


def maximum_health(level, defense):
    return 69 + 3 * level + 28 * defense


def arrow_damage(agility):
    return 2 + agility


def damage_taken(attack_damage, defense):
    return round(attack_damage * (1.2 - 0.01 * defense))


def red_combo_damage(lvl_red, str_red, def_blue):
    strong_damage = damage_taken(strong_attack_damage(lvl_red, str_red),
                                 def_blue)
    normal_damage = damage_taken(normal_attack_damage(lvl_red, str_red),
                                 def_blue)
    return 2 * strong_damage + normal_damage


def blue_combo_damage(lvl_blue, str_blue, def_red):
    normal_damage = damage_taken(normal_attack_damage(lvl_blue, str_blue),
                                 def_red)
    throw_damage = damage_taken(throw_attack_damage(lvl_blue, str_blue),
                                def_red)
    return 3 * normal_damage + throw_damage
```

```python
def fight(lvl_red, str_red, def_red, hp_perc_red,
          lvl_blue, str_blue, def_blue, hp_perc_blue):
    health_red = round(hp_perc_red / 100 * maximum_health(lvl_red, def_red))
    health_blue = round(hp_perc_blue / 100 * maximum_health(lvl_blue, def_blue))

    loss_blue = red_combo_damage(lvl_red, str_red, def_blue)
    loss_red = blue_combo_damage(lvl_blue, str_blue, def_red)

    rounds_to_kill_red = math.ceil(health_red / loss_red)
    rounds_to_kill_blue = math.ceil(health_blue / loss_blue)

    print('It takes red', rounds_to_kill_blue, 'rounds to kill blue.')
    print('It takes blue', rounds_to_kill_red, 'rounds to kill red.')

    if rounds_to_kill_red < rounds_to_kill_blue:
        print('Blue wins!')
    else:
        print('Red wins!')


level_red = 31
strength_red = 20
defense_red = 30
start_health_perc_red = 25

level_blue = 31
strength_blue = 20
defense_blue = 30
start_health_perc_blue = 20

fight(level_red, strength_red, defense_red, start_health_perc_red,
      level_blue, strength_blue, defense_blue, start_health_perc_blue)
```

Output:

```
It takes red 3 rounds to kill blue.
It takes blue 3 rounds to kill red.
Red wins!
```

Alternative Solution

File: code/castle_crashers_alternative_solution.py

```python
def strong_attack_damage(level, strength):
    return (5 + 1.15 * strength + 0.1 * level) // 1
```

```python
def normal_attack_damage(level, strength):
    return (3 + strength + 0.1 * level) // 1


def throw_attack_damage(level, strength):
    return (10 + 1.2 * strength + 0.1 * level) // 1


def maximum_health(level, defense):
    return 69 + 3 * level + 28 * defense


def arrow_damage(agility):
    return 2 + agility


def damage_taken(attack_damage, defense):
    return (attack_damage * (1.2 - 0.01 * defense) + 0.5) // 1


# the attributes of our knights
level = 31
strength = 20
magic = 20
defense = 30
agility = 7

# initializing health values
red_health = round(0.25 * maximum_health(level, defense))
blue_health = round(0.2 * maximum_health(level, defense))

round_counter = 0

# now: FIGHT! until red_health or blue_health are below 0
while red_health > 0 and blue_health > 0:
    # next round!
    round_counter = round_counter + 1

    # red attacks and blue takes damage
    damage_taken_blue = 2 * damage_taken(strong_attack_damage(level, strength), defense) \
                        + damage_taken(normal_attack_damage(level, strength), defense)
    blue_health = blue_health - damage_taken_blue
    print('Round: ', round_counter, ' Blue health: ', blue_health)
    # blue attacks and red takes damage
    damage_taken_red = 3 * damage_taken(normal_attack_damage(level, strength), defense) \
                         + damage_taken(throw_attack_damage(level, strength), defense)
    red_health = red_health - damage_taken_red
    print('Round: ', round_counter, ' Red health: ', red_health)

if blue_health < 0:
    print('Blue goes down first! Red wins!')
else:
    print('Red goes down first! Blue wins')
```

Output:

```
Round:  1  Blue health:  121.0
Round:  1  Red health:  148.0
Round:  2  Blue health:  42.0
Round:  2  Red health:  46.0
Round:  3  Blue health:  -37.0
Round:  3  Red health:  -56.0
Blue goes down first! Red wins!
```