Oscar Orellana González

1. Which paradigms does Python support?
    - Imperative and Functional
    - Functional and Object-Oriented
    - Imperative and Object-Oriented
    - **Imperative, Functional, and Object-Oriented**

2. What is PEP and which is the one that stablish the Style guide for Python?
    - PEP is Python Established Propositions, PEP 9
    - **PEP is Python Enhancement Proposal, PEP 8**
    - PEP is Python Enhancement Production, PEP 8
    - PEP is Proggraming Enhacement for Python, PEP 20

3. What are the pitfalls and problems of Python language?
    - Block Comments, No Prebuilt Statistical Models or Tests
    - **Speed, Memory Consumption, Database Access, Runtime Errors, Difficulty in Using Other Languages, Simplicity**
    - Easy to read, learn and write, Improved productivity, Dynamically Typed, Vast Libraries Support Portability

4. Is it possible to use the construction `True = False`?
    - Yes, True is the name of the variable to which the boolean value of False is assigned.
    - **No,you cannot assign a value to Python keyword "True".**

5. When will the `else` part of `try...except...else` be executed?
    - At the end of each block, try to ... except for
    - After exception occurs
    - **When no exception occurs**
    - Always when the try block fails

6. How are set implemented internally
    - If multiple values are present in the same index position, the value is added to that index position to form a linked list.
    - are implemented using a dictionary with dummy variables, where the key is the set of members with the highest optimizations to time complexity.

- **In Python, sets are created through the set() function. An empty list is created. Note that the empty set cannot be created through {}, it creates a dictionary.**

7. Which is the out for this code?

```
class Parent:
    def met(self):
        print("In Parent")

class ChildOne(Parent):
    def met(self):
        print("In ChildOne")

class ChildTwo(Parent):
    def met(self):
        print("In ChildTwo")

class Weirdo(ChildOne, ChildTwo):
    def met(self):
        print("In Weirdo")
        ChildOne.met(self)
        ChildTwo.met(self)
        Parent.met(self)

obj = Weirdo()
obj.met()
```

-
  ```
  In Weirdo
  In Parent
  In Parent
  In Parent
  ```
-
  In Weirdo
  In ChildOne
  In ChildTwo
  In Parent
-
  ```
  In Parent
  In ChildOne
  In ChildTwo
  In Weirdo
  ```

8. How does the MRO works in Python?
   - MRO searches for sorted classes in a superclass it search from top to bottom and from left to right.

- MRO searches for a method, if it exists is selected from right to left and from top to bottom.
- **MRO searches for the method in an object's class, from bottom to top and left to right.**

9. What happens when you do “print(Weirdo.mro())”:
    - [<class '__main__.Parent'>, <class 'object'>]
    - [<class '__main__.ChildOne'>, <class '__main__.Parent'>, <class 'object'>]
    - [<class '__main__.Weirdo'>, <class '__main__.ChildTwo'>, <class '__main__.ChildOne'>, <class '__main__.Parent'>, <class 'object'>]
    - **[<class '__main__.Weirdo'>, <class '__main__.ChildOne'>, <class '__main__.ChildTwo'>, <class '__main__.Parent'>, <class 'object'>]**

10. What are descriptors
    - **Descriptors are Python objects that implement a method of the descriptor protocol.**
    - Descriptor does not give precise control over access to attributes.
    - a descriptor is an object instance with "binding behavior".

11. From the next code, What is “fund” doing to “function”?

```
import time
def fund(func):
    def funw():
        time.sleep(2.5)
        func()
        print("Done")
    return funw

@fund
def function():
    print("I am a function")

function()
```

- “fund” replaces “function”, adding “Done” on it. and waits 2.5s
- “fund” decorates “function”, waits 2.5s and prints “Done” before called
- **“fund” decorates “function”, waits 2.5s and prints “Done” after calling func() = function() = “I am a function”**
- “funds” waits 2.5s, overrides the print methond in “function” with “Done”

12. How are arguments passed to function in Python — by value or by reference?

- immutable arguments by value
- mutable arguments by reference
- **pass by object reference**

13. Whenever Python exits, why isn't all the memory de-allocated?

- python does not use the garbage collector
- **you will lose memory when you declare circular references and implement a custom __del__ destructor method on one of these classes**
- Python modules are always deallocated.
- Python recognizes and frees circular memory references before using the garbage collector.

14. What is GIL

- **is a mutex (or a lock) that allows only one thread to have control of the Python interpreter.**
- Several threads can be in a running state at any time.
- GIL allows multiple threads to run at the same time, even in a multi-threaded architecture with more than one CPU core.

15. Why does this happen in a python terminal?

```
>>> a = "Yang"
>>> b = "Yang"
>>> a is b
True
>>> c = "Yang Zhou"
>>> d = "Yang Zhou"
>>> c is d
False
>>> d is c
```

- Memory managment, for similar values
- Memory managment, similar registry
- Memory system protocol for registry
- **Chain value for registry in memory allocation**

16. What is __pycache__ and .pyc files?

- **are the compiled and cached bytecode**
- are file that is stored in __pycache__ to be executed.

- are files stored in __pycache__ for code retrieval

17. What's the output?

```
def Foo():
    yield 42;
    return 666
```

- returns nothing
- returns 666
- **returns a generator**
- returns an error

18. The output of the following code is

```
Receptor
message='I am a message'
```

```
from pydantic import BaseModel
from typing import Any

class Message(BaseModel):
...

class Letter(BaseModel):
...

message = Message(**{'message': 'I am a message'})
data= {
    'message': message,
    'receptor': 'Receptor'
}

letter = Letter(**data)

print(letter.receptor)
print(letter.message)
```

How class Message and Letter should look like?

-

```
class Message(BaseModel):
```

```
    message: dict[str, Any]

class Letter(BaseModel):
    message: str
    receptor: str
```

- **This one**

```
class Message(BaseModel):
    message: str

class Letter(BaseModel):
    message: Message
    receptor: str
```

19. How the garbage collector works in Python
    - The Python garbage collector is executed during program compilation.
    - Python deletes unwanted objects (built-in types or class instances) automatically to free up memory space.
    - Python's memory allocation and deallocation method is manual.
    - **The garbage collector keeps track of all objects in memory.**

20. The N queens puzzle is the challenge of placing N non-attacking queens on an N×N chessboard. Write a program that solves the N queens problem.
    - Usage: nqueens N
        i. If the user called the program with the wrong number of arguments, print Usage: nqueens N, followed by a new line, and exit with the status 1
    - where N must be an integer greater or equal to 4
        i. If N is not an integer, print N must be a number, followed by a new line, and exit with the status 1
        ii. If N is smaller than 4, print N must be at least 4, followed by a new line, and exit with the status 1
    - The program should print every possible solution to the problem
        i. One solution per line
        ii. Format: see example
        iii. You don't have to print the solutions in a specific order
    - You are only allowed to import the sys module

```
carbonell@ubuntu:~/N Queens$ ./0-nqueens.py 4
[[0, 1], [1, 3], [2, 0], [3, 2]]
[[0, 2], [1, 0], [2, 3], [3, 1]]
carbonell@ubuntu:~/N Queens$ ./0-nqueens.py 6
[[0, 1], [1, 3], [2, 5], [3, 0], [4, 2], [5, 4]]
[[0, 2], [1, 5], [2, 1], [3, 4], [4, 0], [5, 3]]
[[0, 3], [1, 0], [2, 4], [3, 1], [4, 5], [5, 2]]
[[0, 4], [1, 2], [2, 0], [3, 5], [4, 3], [5, 1]]
```

Nqueens.py

```
import sys


def _get_arg(argv):
    """
    Get argument if it is valid or exits the program
    """
    # Wrong number of arguments
    if len(argv) != 2:
        print("Usage: nqueens N")
    # N must be an integer greater or equal to 4
    else:
        if argv[1].isdigit():
            number = int(argv[1])
            if number > 3:
                return number
            else:
                print("N must be at least 4")
        else:
            print("N must be a number")
    sys.exit(1)


def _build_chessboard():
    """
    Builds the chessboard taking N to create the dimension.
    """
    return [[" " for i in range(n)] for j in range(n)]


def _print_solution():
    """
    Given a solution it is printed accordingly to the format [[column,
row],[column, row],[column, row],[column, row]]
    indicating where are the queens.
    """
    solution = []
    for i in range(n):
        solution.append([i, chessboard[i].index("X")])
    print(solution)


def _is_safe_same_upper_col(row, col):
    """
    Check if there is any queen for same upper column
    """
```

```
    while row >= 0:
        if chessboard[row][col] == "X":
            return False
        else:
            row -= 1
    return True


def _is_safe_upper_right_diagonal(row, col):
    """
    Check if there is any queen for upper right diagonal
    """
    while col < n and row >= 0:
        if chessboard[row][col] == "X":
            return False
        else:
            col += 1
            row -= 1
    return True


def _is_safe_upper_left_diagonal(row, col):
    """
    Check if there is any queen for upper left diagonal
    """
    while col >= 0 and row >= 0:
        if chessboard[row][col] == "X":
            return False
        else:
            row -= 1
            col -= 1
    return True


def _is_safe(row, col):
    """
    Check if there is safe to locate a queen in a specific cell
    """
    if _is_safe_same_upper_col(row, col) is False:
        return False
    if _is_safe_upper_right_diagonal(row, col) is False:
        return False
    if _is_safe_upper_left_diagonal(row, col) is False:
        return False
    return True


def _solve_n_queens(row):
    """
    Main function to solve n queens.

    Queen is depicted by "X". The strategy solves 1 case and rest
recursion will follow. For each position, it checks if
    it is safe and if it is safe it makes a recursive call with row+1,
chessboard[i][j]='X' and then revert the change
    in the chessboard that is make the chessboard[i][j]=' ' again to
generate more solutions
    """
    if row == n:
        _print_solution()
        return
```

```
    for col in range(n):
        if _is_safe(row, col):
            chessboard[row][col] = "X"
            _solve_n_queens(row + 1)
            chessboard[row][col] = " "


n = _get_arg(sys.argv)
chessboard = _build_chessboard()
_solve_n_queens(0)
```

21. Create a function def pascal_triangle(n): that returns a list of lists of integers representing the Pascal's triangle of n:
    - Returns an empty list if n <= 0
    - You can assume n will be always an integer

```
carbonell@ubuntu:~/pascal$ cat 0-main.py
#!/usr/bin/python3
"""
0-main
"""
pascal_triangle = __import__('0-pascal_triangle').pascal_triangle

def print_triangle(triangle):
    """
    Print the triangle
    """
    for row in triangle:
        print("[{}]".format(",".join([str(x) for x in row])))


if __name__ == "__main__":
    print_triangle(pascal_triangle(5))

carbonell@ubuntu:~/pascal$
carbonell@ubuntu:~/pascal$ ./0-main.py
[1]
[1,1]
[1,2,1]
[1,3,3,1]
[1,4,6,4,1]
```

Pascal_triangle.py

```
def pascal_triangle(number: int):
    """
    Pascal's Triangle is a kind of number pattern. Pascal's Triangle is
the triangular arrangement of numbers that
```

```
    gives the coefficients in the expansion of any binomial expression.
This function follows the concept of a
    Binomial Coefficient. The idea is to calculate C(line, i) using
C(line, i-1) in all lines. ->
    C(line, i) = C(line, i-1) * (line - i + 1) / i.
    """
    output = []
    if number > 0:
        for i in range(1, number + 1):
            c = 1
            b = []
            for j in range(1, i + 1):
                b.append(c)
                c = c * (i - j) // j
            output.append(b)
    return output
```

22. Maria and Ben are playing a game. Given a set of consecutive integers starting from 1 up to and including n, they take turns choosing a prime number from the set and removing that number and its multiples from the set. The player that cannot make a move loses the game.
They play x rounds of the game, where n may be different for each round. Assuming Maria always goes first and both players play optimally, determine who the winner of each game is.

- Prototype: def isWinner(x, nums)
- where x is the number of rounds and nums is an array of n
- Return: name of the player that won the most rounds
- If the winner cannot be determined, return None
- You can assume n and x will not be larger than 10000
- You cannot import any packages in this task

Example:

- x = 3, nums = [4, 5, 1]

First round: 4

- Maria picks 2 and removes 2, 4, leaving 1, 3
- Ben picks 3 and removes 3, leaving 1
- Ben wins because there are no prime numbers left for Maria to choose

Second round: 5

- Maria picks 2 and removes 2, 4, leaving 1, 3, 5
- Ben picks 3 and removes 3, leaving 1, 5
- Maria picks 5 and removes 5, leaving 1
- Maria wins because there are no prime numbers left for Ben to choose

Third round: 1

- Ben wins because there are no prime numbers for Maria to choose

Result: Ben has the most wins

```
carbonell@ubuntu:~/primegame$ cat main_0.py
#!/usr/bin/python3

isWinner = __import__('0-prime_game').isWinner

print("Winner: {}".format(isWinner(5, [2, 5, 1, 4, 3])))

carbonell@ubuntu:~/primegame$ ./main_0.py
Winner: Ben
```

primegame.py

```
def is_winner(x, nums):
    """
    Function that returns the winner of the prime game
    """
    wins_maria, wins_ben = 0, 0
    # Play the rounds
    for i in range(x):
        prime_numbers = _get_amount_prime_numbers(nums[i])
        if prime_numbers == 0:
            pass
        elif prime_numbers % 2 == 0:
            wins_ben += 1
        else:
            wins_maria += 1
    # Assess the winner
    if wins_ben == wins_maria:
        return None
    return "Ben" if wins_ben > wins_maria else "Maria"


def _get_amount_prime_numbers(num):
    """
    Given a number returns the amount of prime numbers from 2 to
that number
    """
    total_prime_numbers = 0
    for i in range(2, num + 1):
        is_prime = True
        for j in range(2, i // 2 + 1):
            if i % j == 0:
                is_prime = False
                break
        if is_prime:
            total_prime_numbers += 1

    return total_prime_numbers
```