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

1. 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”.**

1. 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

1. 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.**
2. 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 |

1. 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.**

1. 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'>]**

1. 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".
2. 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”

1. 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**
2. 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.
3. 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.
4. 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**

1. 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
2. What's the output?

|  |
| --- |
| def Foo():  yield 42;  return 666 |

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

1. 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 |

1. 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.**
2. 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
     1. 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
     1. If N is not an integer, print N must be a number, followed by a new line, and exit with the status 1
     2. 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
     1. One solution per line
     2. Format: see example
     3. 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]] |

1. 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] |

1. 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 |