**Exam 2 – Paper and Pencil part (Fall, 2020-21)**

**Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Section: \_\_\_\_\_\_\_**

***Rules and Expectations*** *Sections 1 and 2 for Mutchler. Section 3 for Noureddine. Section 4 for Ebrahimi.*

At the beginning of this exam, you will receive the **Expectations about Academic Integrity** for this exam -- it is the same as what you were given to read previously. Re-read that document as needed. ***Sign it and turn it in when you finish this exam (both parts).***

***Two parts (this is Part 1, Paper-and-Pencil)***

**For this part, the ONLY external resource you may use is *two* 8½ by 11-inch sheets of paper** (or ***one double-sided sheet***), with whatever you want on it, typed or handwritten or a combination of the two. You must have prepared the sheets *before* beginning this exam. You may also use a calculator if you like (but only for calculating).

|  |  |  |  |
| --- | --- | --- | --- |
| **Problem** | **Points Possible** | **Points Earned** | **Comments** |
| **1** | **7** |  |  |
| **2** | **3** |  |  |
| **3** | **10** |  |  |
| **4** | **10** |  |  |
| **5** | **8** |  |  |
| **6** | **12** |  |  |
| **Total** (of 100 on the exam) | **50** |  |  |

***Communication***

For both parts of the exam, ***you must not communicate with anyone*** except your instructors and their assistants, if any. In particular:

* You must not talk with anyone else or exchange information with them during this exam.
* ***After*** this exam, **you must not talk about the exam with anyone who has not yet taken it.**

***Do NOT use email***, *chat* or the like during this exam. **Close any such applications now.**

1. Consider a function named ***do\_it*** that takes a ***list of integers*** as its sole argument. For each of the following possible specifications for what ***do\_it*** returns:

Circle ***Yes*** if the code for ***do\_it***  would require a loop.

Circle ***No***  if the code for ***do\_it***  would NOT require a loop.

If ***do\_it*** returns:

* 1. The number of odd integers in the list. **Yes No**
  2. The average of the integers in the list. **Yes No**
  3. The number of integers in the list. **Yes No**
  4. ***True***  if the first integer in the list is even, else ***False***. **Yes No**
  5. ***True***  if the list contains no even integers, else ***False***. **Yes No**
  6. The second smallest integer in the list. **Yes No**
  7. The last integer in the list. **Yes No**

1. Consider a function named ***middle*** that takes a single argument:  
   a sequence of numbers, where the length of the sequence is *guaranteed* to be odd.

The function returns the number at the middle of the sequence. For example:

* middle( [90, **10**, 45] ) returns **10**
* middle( [4, 1, 6, **18**, 10, 12, 21] ) returns **18**
* middle( [**33**] ) returns **33**

|  |
| --- |
| def middle(numbers):  def middle(numbers):  return numbers[len(numbers) // 2] |

Write (in the box below) a complete implementation of the above ***middle*** function.

1. Consider the code snippet below. It is a contrived example with poor style, but it will run without errors. What does it print when it runs? Write your answer in the box to the right of the code. Show your work by making notations in the code or by using the empty space below or on another sheet of paper, as desired.

|  |  |
| --- | --- |
| def home():  x = bear(3, 8)  print("Home A:", x)  y = bear(ant(4), ant(1))  print("Home B:", y)  def ant(x):  print("Ant:", x)  x = x + 5  return x  def bear(x, y):  print("Bear A:", x, y)  r = ant(x + 4)  print("Bear B:", r)  return ant(r + y)  print(ant(100))  return 33  print("OK:", ant(101) % 100)  home() | **Output**: |

1. Consider the code on the next page. It is a contrived example with poor style but will run without errors. In this problem, you will trace the execution of the code. As each location is encountered during the run:

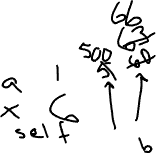
* ***CIRCLE* each variable** that is ***defined*** at that location.
* ***WRITE*** the ***VALUE*** of each variable that you ***circled*** directly ***BELOW*** the circle.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Location 1** *(1st time)* | **a** | **b** | **x** | **self.a** | **self.b** | **bear.a** | **bear.b** | **cat.a** | **cat.b** | **dog.a** | **dog.b** |
| **Location 1** *(2nd time)* | **a** | **b** | **x** | **self.a** | **self.b** | **bear.a** | **bear.b** | **cat.a** | **cat.b** | **dog.a** | **dog.b** |
| **Location 2** *(1st time)* | **a** | **b** | **x** | **self.a** | **self.b** | **bear.a** | **bear.b** | **cat.a** | **cat.b** | **dog.a** | **dog.b** |
| **Location 2** *(2nd time)* | **a** | **b** | **x** | **self.a** | **self.b** | **bear.a** | **bear.b** | **cat.a** | **cat.b** | **dog.a** | **dog.b** |
| **Location 3** *(1st time)* | **a** | **b** | **x** | **self.a** | **self.b** | **bear.a** | **bear.b** | **cat.a** | **cat.b** | **dog.a** | **dog.b** |
| **Location 3** *(2nd time)* | **a** | **b** | **x** | **self.a** | **self.b** | **bear.a** | **bear.b** | **cat.a** | **cat.b** | **dog.a** | **dog.b** |
| **Location 4** | **a** | **b** | **x** | **self.a** | **self.b** | **bear.a** | **bear.b** | **cat.a** | **cat.b** | **dog.a** | **dog.b** |
| **Location 5** | **a** | **b** | **x** | **self.a** | **self.b** | **bear.a** | **bear.b** | **cat.a** | **cat.b** | **dog.a** | **dog.b** |
| **Location 6** | **a** | **b** | **x** | **self.a** | **self.b** | **bear.a** | **bear.b** | **cat.a** | **cat.b** | **dog.a** | **dog.b** |
| **Location 7** | **a** | **b** | **x** | **self.a** | **self.b** | **bear.a** | **bear.b** | **cat.a** | **cat.b** | **dog.a** | **dog.b** |
| **Location 8** | **a** | **b** | **x** | **self.a** | **self.b** | **bear.a** | **bear.b** | **cat.a** | **cat.b** | **dog.a** | **dog.b** |

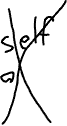
|  |
| --- |
| class Animal():  def \_\_init\_\_(self, a, x):  self.a = a + 4  self.b = x \* 10  a = 31  #### --- Location 1 ---  def zoo(self, a, bear):  #### --- Location 2 ---  a = a + 1  self.a = self.a + 100  bear.b = bear.b + 3  #### --- Location 3 ---  return self.b + bear.a  def beach(b):  #### --- Location 4 ---  return Animal(b, b + 5)  Loc 3 (2nd time): a 3 self.a 105 self.b 66 bear.a 105 bear.b 66  def main():  a = 10  b = 8  cat = Animal(a, b)  #### --- Location 5 ---  dog = beach(1)  #### --- Location 6 ---  x = cat.zoo(1, dog)  #### --- Location 7 ---  a = dog.zoo(2, dog)  #### --- Location 8 ---  main() |

dddcc

Loc 8: a 171 b 85 cat.a 114 cat.b 80 dog.a 105 dog.b 66



Loc 2 (2nd time): a 2 self.a 5 self.b 63 bear.a 5 bear.b 63



Loc 7: x 85 a 10 b 8 cat.a 114 cat.b 80 dog.a 5 dog.b 63

Loc 2: a 1 self.a 14 self.b 80

bear.a 5 bear.b 60

Loc 5: a 10 b 8 cat.a 14 cat.b 80

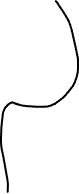
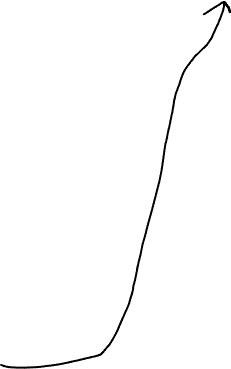
Loc 3: a 2 self.a 114 self.b 80 bear.a 5 bear.b 6

Loc 4 b 1



Loc 1 (2nd time): a 31 x 6 self.a 5 self.b 60

Loc 6: a 10 b 8 cat.a 14 cat.b 80  
dog.a 5 dog.b 60



Location 1: a 31 x 8 self.a 14 self.b 80



1. Consider the code snippet below. It is a contrived example with poor style, but it will run without errors.

What does it print when it runs? Write your answer in the box below.

Show your work in any way that you think would be helpful.

|  |  |
| --- | --- |
| s = [5, 3, 1, 8, 4,  9, 7, 6, 2, 10, 20]  a = 3  b = 0  print(len(s))  for k in range(1, 10, 3):  a = a + s[k]  b = b + s[len(s) - k - 1]  print("A.", k, len(s) - k - 1)  print("B.", s[k], s[len(s) - k - 1])  print("C.", a, b)  print()  print("Now:", a, b) |  |

1. Consider a function named ***average*** that takes a single argument:  
   a sequence of numbers, where the length of the sequence is *guaranteed* to be odd.

The function returns the average of the numbers in the sequence that are bigger than or equal to the number at the middle of the sequence. For example:

* average( [1, 7, 6, 5, 10, 3, 9] ) returns **7.4**

since the number at the middle of the sequence is **5**

and the numbers in the list bigger than or equal to **5** are  **7**, **6**, **5**, **10** and **9**

and those **5** numbers add up to  **7 + 6 + 5 + 10 + 9**, which is **37,**

and the average of **5** numbers that sum to **37** is **37 / 5**, which is **7.4**.

* average( [2, 1, 4, 4, 7] ) returns **5.0** since the number at the middle of the sequence is **4,** and the numbers in the list bigger than or equal to **4** are **4, 4** and **7**, and the sum of those **3** numbers is **15**, so the average of them is **15/3**, which is **5.0**.
* average( [9, 4, 7, 6, 2] ) returns **8.0** since the number at the middle of the sequence is **7,** and the numbers in the list bigger than or equal to **7** are **9** and **7**, and the sum of those **2** numbers is **16**, so the average of them is **16/2**, which is **8.0**.

Write (in the box below) a complete implementation of the above ***average*** function.

|  |
| --- |
| def average(numbers): |

This is the back page of this exam. Use it for scratch work if you like.