An Introduction To Interactive Programing In Python (Part 2)

by Joe Warren, Scott Rixner, John Greiner, Stephen Wong

Quiz 6b – Tiled images *Skanda S Bharadwaj*

Question 1

What is the position of the center of the top-left card (Ace of Clubs, A♣) in the tiled image discussed in the "Tiled images" video? Remember that each card in this tiled image has size 73 x 98 pixels.

(Note that the tiled image used in the current version of your Blackjack mini-project is slightly smaller.)

Your Answer		Score	Explanation
`(73, 98)`			
`(5 * 73 + 36.5, 1 * 98 + 49)`			
`(0, 0)`			
✓ `(36.5, 49)`	Correct	10.00	

Total	10.00 / 10.00

Question 2

What is the position of the center of the bottom-right card (King of Diamonds, K♦) in the tiled image discussed in the "Tiled images" video? Again, remember that each card in this tiled image has size 73 x 98 pixels.

Enter two numbers, separated only by spaces.

Answer for Question 2

You entered:

Your Answer		Score	Explanation
✓ 912.5	Correct	5.00	
✓ 343.0	Correct	5.00	
Total		10.00 / 10.00	

Question 3

When using Dropbox to store images for use with CodeSkulptor, what should the `www` portion of the DropBox URL be replaced by?

Refer to the video on tiled images.

Your Answer		Score	Explanation
✓ `dl`	Correct	10.00	Yes. `dl` specifies that the file is for download.
`www`			
`gif`			
`html`			
`jpg`			
Total		10.00 / 10.00	

Question 4

Within the `__init__` method, the new object should be returned with what code?

Your Answer		Score	Explanatio
✓ No `return` statement is needed in `__init__`.	Correct	10.00	
`return self`			
`return whatever_the_object_is_named` (Use the appropriate variable name.)			
`return`			
Total		10.00 / 10.00	

Question Explanation

Here are some hidden details to explain this potentially confusing behavior. Each Python class has a hidden *constructor* method that

- constructs (makes) the object,
- calls `__init__` to initialize the object,
- then returns this object.

So, while there is a `return` statement somewhere, it is in this hidden constructor method that you don't have to define.

Question 5

One way of understanding code is to think about other code that accomplishes the same thing — i.e., given the same starting values, it returns and/or mutates the same values.

This following defines one way to concatenate multiple lists. For example, `list_extend_many([[1,2], [3], [4, 5, 6], [7]])` returns `[1, 2, 3, 4, 5, 6, 7]` and doesn't mutate anything.

```
def list_extend_many(lists):
    """Returns a list that is the concatenation of all the lists in the given list-of-lists."""
    result = []
    for l in lists:
        result.extend(l)
    return result
```

Which of the following definitions are equivalent? I.e., which always produce the same output for the same input, and never mutate the input or any global variable?

Your Answer		Score	Explanation

def list_extend_many(lists):
result = []
i = len(lists)
while i >= 0:
	Correct	1.00	This loops over all the items in the list, but in the reverse order. Sometimes that is fir
			but here it the result is in reversed.

i -= 1 result.extend(lists[i]) return result			
✓ def list_extend_many(lists): result = [] i = 0 while i < len(lists): result.extend(lists[i]) i += 1 return result	Correct	4.00	
✓ def list_extend_many(lists): result = [] i = 0 while i < len(lists): result += lists[i] i += 1 return result	Correct	4.00	
def list_extend_many(lists): result = []	Correct	1.00	This returns the correct result, but it mutates the argument in the process! So, if you have some code like the following:

```
    while len(lists) > 0:
        result.extend(lists.pop(
0))
    return result
```

```
lists = [[1 2], [3], [4, 5, 6], [7]]

print list_extend_many(lists)

print lists
```

It would show that `lists` is empty at the end.

Total	10.00 / 10.00

Question 6

Which of the following programs would never end if it weren't for CodeSkulptor's timeout? Assume no `break` or `return` statement is used in the elided loop bodies.

You might want to add a `print` statement to each loop to better understand the behavior.

Your Answer		Score	Explanation

n = 1000
while n > 0:
    …   # Assume this doesn't modify n.
    n -= 1
	Correct	1.00	

✓ n = 1 while n > 0: … # Assume this doesn't modify n. n += 1	Correct	4.00
my_list = … for x in my_list: … # Assume this doesn't mutate my_list.	Correct	1.00
✓ n = 1001 while n != 0: … # Assume this doesn't modify n. n -= 2	Correct	4.00
Total		10.00 / 10.00

Question 7

Convert the following English description into code.

1. Initialize `n` to be 1000. Initialize `numbers` to be a list of numbers from 2 to n, but not including `n`.
2. With `results` starting as the empty list, repeat the following as long as `numbers` contains any numbers.
   - Add the first number in `numbers` to the end of `results`.

- Remove every number in `numbers` that is evenly divisible by (has no remainder when divided by) the number that you had just added to `results`.

How long is `results`?

To test your code, when `n` is instead 100, the length of `results` is 25.

Answer for Question 7

You entered:

Your Answer		Score	Explanation
✓ 168	Correct	20.00	Correct. This computes the primes less than `n` by a process known as the Sieve of Eratosthenes. By the way, here's one way to write this:

```
n = 1000
numbers = range(2, n)
results = []


while numbers != []:
    results.append(numbers[0])
    numbers = [n for n in numbers if n % numbers[0] != 0]
```

		`print len(results)`
Total	20.00 / 20.00	

Question 8

We can use loops to simulate natural processes over time. Write a program that calculates the populations of two kinds of "wumpuses" over time. At the beginning of year 1, there are 1000 slow wumpuses and 1 fast wumpus. This one fast wumpus is a new mutation. Not surprisingly, being fast gives it an advantage, as it can better escape from predators. Each year, each wumpus has one offspring. (We'll ignore the more realistic niceties of sexual reproduction, like distinguishing males and females.). There are no further mutations, so slow wumpuses beget slow wumpuses, and fast wumpuses beget fast wumpuses. Also, each year 40% of all slow wumpuses die each year, while only 30% of the fast wumpuses do.

So, at the beginning of year one there are 1000 slow wumpuses. Another 1000 slow wumpuses are born. But, 40% of these 2000 slow wumpuses die, leaving a total of 1200 at the end of year one. Meanwhile, in the same year, we begin with 1 fast wumpus, 1 more is born, and 30% of these die, leaving 1.4. (We'll also allow fractional populations, for simplicity.)

Beginning of Year	Slow Wumpuses	Fast Wumpuses
1	1000	1
2	1200	1.4

3	1440	1.96
...	...	...

Enter the first year in which the fast wumpuses outnumber the slow wumpuses. Remember that the table above shows the populations at the start of the year.

Answer for Question 8

You entered:

Your Answer		Score	Explanation
✓ 45	Correct	20.00	
Total		20.00 / 20.00	