Midterm notes

q1: Looping

```python
def q1():
    x = 5
    y = 2
    while y < x :
        x += 1
        y += 2
    print(y - x)
```

I wanted to see if you can reason about series of variable values in a loop.

Most common error: One too many or one too few loop iteration.

```
x  5  6  7  8  
y  2  4  6  8  
```

From the pattern we can see the loop will end when x==y, so it will print 0

q2: Method call, modular arithmetic, more complex code

```python
def add_time(start, elapsed):
    hh = start // 100
    mm = start % 100
    el_hh = elapsed // 100
    el_mm = elapsed % 100
    if mm >= 60 :
        hh += 1
        mm = mm - 60
    hh = hh + el_hh
    if hh > 12 :
        hh = hh % 12
    return 100*hh + mm

def q2():
    t = add_time(1130, 245)
    print(t)
```

We used integer division and remainder to break up numbers in the alphacode project. I wanted to know if you could trace execution through a more complex sequence of steps, including a procedure call.

Most common error: Forgetting how integer division (//) and remainder (%) work.

q3: Procedure call mechanism

```python
def f(x, y):
    return x + y

def f2(x):
    return f(x, x)

def q3():
    x = 4
    y = 7
    x = f(x, y)
    return f(x, y)
```

I wanted to check your understanding of function calls, scope and parameter passing.

Most common error: Using the old value of x in the second call to f.

First call to f returns 6; second call is f(7,6) and returns 2.
q4: Nested loops, reasoning about a pattern

```python
def q4():
    ev_count = 0
    for i in [2, 4, 6, 8, 10):
        for j in [1, 2, 3, 4, 5]:
            sum = i + j
            if sum % 2 == 0:
                ev_count += 1
    return ev_count
```

I wanted you to reason about the series of values in a loop without tracing the whole thing (25 iterations).

Each combination of [2, 4, 6, 8, 10] with [1, 5] is checked. But only the even sums are counted. Since even + even = even, even + odd = odd, the combinations that matter are [2, 4, 6, 8, 10] with [2, 4]: 10 combinations.

Most common error: Misunderstanding nested loops, pairing only 2+1, 4+2, 6+3, 8+4, 10+5 instead of all 25 pairs.

---

q5: Counting loop, function

I wanted to see if you could design a function for part of a task and then use it, and if you understood docstring comments as function specifications.

Some of you used str.count(HASH) ... ok. (I forgot about it.)

Some of you repeated the counting loop in is_spam_hash (tweet). Not ok.

Some didn’t follow the docstring comment specification. Not ok.

Several errors in scope (using a local variable of num_hash within is_spam_hash and calling the num_hash function).

---

q6: Loop design

Challenging question: Can you design a loop over pairs of adjacent elements, comparing them.

Requires a combination of two techniques:

• Loop over pairs of adjacent elements  
  (three approaches ... see next slide)

• Return true if all pairs meet the test, false if any pair fails

---

Looping over pairs

```python
for i in range(len(ar)-1):
    prev = ar[i]
    cur = ar[i+1]
    diff = cur - prev
    do something
```

I also saw combinations: Looping with “for i in ar: “ while keeping count of position. Possible but clumsy.

Common errors: Nested loops, confusion between looping through indexes and looping through elements.
**all x in collection are green:**

![Diagram showing code snippets for yes and no cases]

- **Yes**
  ```python
  for x in collection:
    if x is not green:
      return False
  return True
  ```

- **No**
  ```python
  for x in collection:
    if x is green:
      return True
    else:
      return False
  ```

- **Ok**
  ```python
  result = True
  for x in collection:
    if x is not green:
      return False
  return result
  ```

Other possibilities are mostly variations on these. I saw a lot of the “No” pattern above.

---

**Overall impressions**

The exam was fairly difficult

Nonetheless there were a lot of very good scores

We might need more review on
- Loop patterns
- Nested loops
- Functions calls and scope

Don’t look at only your score ... look at what problems you had, and need to work on.