

fullName:_____ andrewID:_____ section:_____

15-112 S25

Quiz1

This is the paper version of Quiz1. You **must write your name on this paper and hand this back** in immediately after the assessment, even if you do not use it to write your answers. If we do not receive it immediately, you will receive a zero on the assessment.

Quiz1 is located in section 1.6 of CS Academy (the end of Unit 1), and contains two parts. Part 1 will become visible once you are allowed to begin. Once the time for Part 1 has elapsed, we will close Part 1 and open Part 2.

We will not grade anything you write on these pages unless you initial the box below:

_____ **Write your initials here if and only if you wish for us to grade what you have written on this paper *instead of* any answers submitted in CS Academy.** Leave the space blank if you wish for us to grade what is written in CS Academy (the default, recommended option). Note: Even if you write your answers on paper, you may not run any code related to the Part 1 problems (MCs and CTs).

You may not view any other notes, prior work, websites or resources, including any form of AI. You may not communicate with anyone else except for TAs or faculty during the assessment. All syllabus policies apply.

Some students will take this at a different time with testing accommodations. As such, you may not discuss this test with anyone else, even briefly, in any form, until we have released grades. Failure to abide by these rules may result in an academic integrity violation.

Do not use strings, loops, sets, dictionaries, recursion, or anything else disallowed in the original problem.

Do not open this or look inside (even briefly) before we instruct you to begin. Close it once you are done.

Part 1[50pts total]: Multiple Choice and Code Tracing

MC1[5 pts]:What will happen if you run the following code: `print(type(int('3.2 + 4.5')))`

- ☐ a. It will print `<class 'int'>`
- ☐ b. It will print `<class 'str'>`
- ☐ c. It will print `<class 'float'>`
- ☐ d. It will crash due to a runtime error

MC2[5 pts]:What will happen if you run the following code: `print(0.1 + 0.1 + 0.1 == 0.3)`

- ☐ a. It will print True
- ☐ b. It will print False
- ☐ c. It will crash due to a syntax error
- ☐ d. It will crash due to a runtime error

MC3[5 pts]:What will happen if you run the following code: `print((30 < 14) and (1/0 != 0))`

- ☐ a. It will print True
- ☐ b. It will print False
- ☐ c. It will crash due to a syntax error
- ☐ d. It will crash due to a runtime error

MC4[5 pts]:Which of the following statements is False?

- ☐ a. The `print(x)` function will display the value of `x`.
- ☐ b. If we write a function that does not include a `return`, that function will return `None` by default.
- ☐ c. The returned value of a function will always be printed.
- ☐ d. When Python executes a `return` statement, it immediately leaves the function call
- ☐ e. If we define a function `f(x)`, we can call it multiple times if we wish.

MC5[5 pts]:What will happen if you run the following code:

```
if 5 - 5:  
    print('a')  
else:  
    print('b')
```

- ☐ a. It will print 'a'
- ☐ b. It will print 'b'
- ☐ c. It will crash due to a syntax error
- ☐ d. It will crash due to a runtime error

CT1[10 pts]:

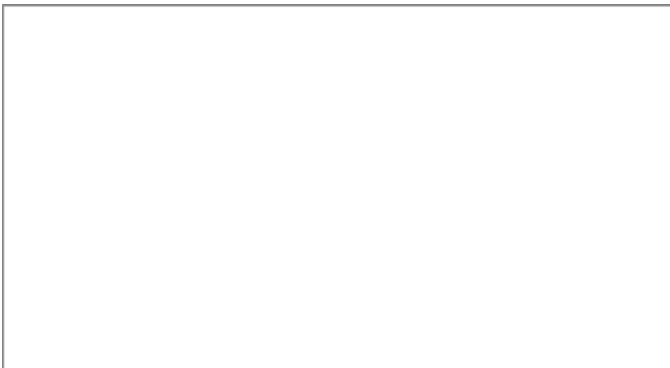
Indicate what the following code prints. Place your answer (and nothing else) in the box below.

```
def f(x):  
    return x + 2  
  
def ct(x, y):  
    z = f(x) + f(y)  
    return f(z)  
  
print(ct(4, 7))
```

A large, empty rectangular box with a thin black border, intended for the user to write the output of the code.**CT2[10 pts]:**

Indicate what the following code prints. Place your answer (and nothing else) in the box below.

```
def ct(x, y):  
    print(x, y%10, y//10)  
    x += 5  
    y = x//10  
    print(x)  
    return y  
  
x = 23  
print(ct(x+1, 45))  
print(x)
```

A large, empty rectangular box with a thin black border, intended for the user to write the output of the code.

Part 2[50pts total]: Free Response

Your functions should work generally for the kinds of inputs specified in the problem statement, and we may test your code using additional test cases. We will manually grade both of these problems for partial credit if you do not pass all the test cases.

FR1[25pts]: getKthDigit(n, k)

Write the function `getKthDigit(n, k)` that takes a possibly-negative int `n` and a non-negative int `k` and returns the `k`th digit in `n`. The 0th digit is the rightmost digit, and counting goes right to left.

Note: In addition to the restrictions on concepts we have not yet covered, **you may not use loops (or strings, or anything else CS Academy would block)** in this problem. Solutions that use loops or strings or uncovered content will receive zero credit.

And here are some test cases:

```
def testGetKthDigit():
    assert(getKthDigit(809, 0) == 9)
    assert(getKthDigit(809, 1) == 0)
    assert(getKthDigit(809, 2) == 8)
    assert(getKthDigit(809, 3) == 0)
    assert(getKthDigit(0, 3) == 0)
    assert(getKthDigit(1, 0) == 1)
    assert(getKthDigit(1, 1) == 0)
    assert(getKthDigit(-1, 0) == 1)
    assert(getKthDigit(-1, 1) == 0)
    assert(getKthDigit(-809, 0) == 9)
```

Define `getKthDigit(n, k)` here

FR2[25 pts]: isPerfectSquare(n)

A perfect square is any non-negative number that can be formed by the product of an integer with itself. For example, $3 * 3 == 9$, so 9 is a perfect square. Also, $5 * 5 == 25$, so 25 is a perfect square. Write the function `isPerfectSquare(n)` which takes an integer `n` (not necessarily positive) and returns `True` if it is a perfect square, and `false` otherwise.

Note: In addition to the restrictions on concepts we have not yet covered, you may not use loops (or strings, recursion, or anything else CS Academy would block) in this problem. **Solutions that use loops will lose 5 points** (but if you aren't sure how to do it without loops, this deduction might be worth it). **Solutions that use other blocked content (like recursion) will receive zero points.**

Hint: If `n` is a perfect square, what do we know about $n^{0.5}$ (i.e. what do we know about its square root?)

Here are some test cases:

```
def testIsPerfectSquare():
    assert(isPerfectSquare(0) == True)
    assert(isPerfectSquare(1) == True)
    assert(isPerfectSquare(2) == False)
    assert(isPerfectSquare(9) == True)
    assert(isPerfectSquare(121) == True)

    # Negative numbers are not perfect squares
    assert(isPerfectSquare(-9) == False)

    #Some very very large numbers
    assert(isPerfectSquare(100000000**2) == True)
    assert(isPerfectSquare(100000000**2 - 1) == False)
```

Define `isPerfectSquare(n)` here or on the following page

#Begin or continue your answer here: