Com S 127
Fall 2016
Topics and more review problems for Exam 1
Monday, October 3, in class

General information

This will be a 50-minute, timed, pencil-and-paper written exam. No books, no notes, no electronic devices, no headphones, no collaboration. The problems will primarily involve writing Python code or reading and interpreting Python code.

Summary of Exam topics

The exam covers everything we have done through Friday, September 30. (You will not be tested on the turtle module.) This corresponds to the first 9 chapters of the textbook (*Eels in my hovercraft*). The list below is a rough summary.

- expressions and statements
- variables and assignment
- calling functions
- arguments and parameters
- calling functions in a module
- conditional statements
- nested conditionals, multiple branches with `elif`
- boolean expressions and boolean operators
- defining functions
- local variables and scope
- problem-solving strategies
- reading and tracing code, drawing a memory map

How do I study for an exam like this?

(Excerpted from Lab 5)

You can become a good at problem-solving and coding with practice and experience.

Given a problem, think of a concrete example to be a "test case". That is, if you wrote the code and wanted to know whether it was correct, how would you check? Often this involves some hand calculation, and often the hand calculation shows you how to write the code. Try to describe to yourself in words the steps in solving the problem.

After writing the code on paper, try to read it. Does it make sense to you? Trace through it by hand and make sure it is really doing what you intended at every step.

Type up your solution in Wing, activecode, or Python Tutor and try it using your concrete examples as test cases.

If you weren't successful, get some help from the TA or Piazza or a fellow student. But ask yourself: In the future, how can I remember the way to solve this kind of problem? How can I recognize a similar problem? Can I make up more problems that are like this one? What information can I stash in my brain to make it easier the next time?

Test-taking also involves some strategy. Ask yourself: What kind of problem am I being asked to solve? Am I reading existing code, answering a question, writing a few statements, writing a function, or what? Am I printing output? Am I reading input? If I am writing a function, what are its parameters? Does it return a value, or does it perform some other action (like drawing or printing)?

Problems to practice on

The main set of exam practice problems was in Lab 5:
http://web.cs.iastate.edu/~smkautz/cs127f16/labs/lab5/lab5_problems.pdf

And be sure you can do all the quizzes:
http://web.cs.iastate.edu/~smkautz/cs127f16/examples/quizzes/

The interactive site codingbat.com is another entertaining way to practice writing code. The problem sets Warmup-1, Logic-1 and Logic-2 are probably relevant for us.
http://codingbat.com/python

There are a few more sample questions below.

Are there answers?

Try your very hardest to read your own code and figure out whether it's correct. After all, that's what you have to do on the exam! You can run the code in Wing 101, activecode windows, or in Python Tutor to check. Take advantage of Piazza - you can post your questions when you're stuck, ask for hints, or post your own code for other students and the instructors to comment on.

Be the professor!

Put yourself in the professor's shoes. Imagine you are writing the exam. What would you put on it? What have we actually covered in the class? Come up with some questions that are not too easy, not too hard, that would test the skills we've practiced. Then make sure you can do them.

A few more problems...

1) For each snippet of code, identify the output, or if there is an error, state what is going wrong.

a)

```
a = 42
b = a
a = a + 1
print(b)
```

b)

```
x = 42
x + 1
print(x)
```

c)

```
def foo(x):
    x = x + 1
    print x

x = 42
foo(x)
print(x)
```

d)

```
def print_area(length, width):
    area = length * width
    print(area)

int(input("Enter length: "))
int(input("Enter width: "))
print_area(length, width)
```

e)

```
def print_area(length, width):
    area = length * width
    print(area)

length = int(input("Enter length: "))
width = int(input("Enter width: "))
print_area()
```

2) For each snippet of code, identify the output, or if there is an error, state what is going wrong.

3) What's the difference between an *argument* and a *parameter* for a function? (Explain briefly and give an example of each.)

4) Explain why in Python, y = x doesn't mean the same thing as x = y.

5) Assume that you have a function and variable defined as follows:

```
def fred(george):
  return george % 3 == 0

hermione = 10
```

Determine the value of each boolean (logical) expression:

```
hermione > 0
hermione > 0 and 25 <= hermione
hermione  <= 25 or hermione < 0
fred(hermione)
hermione > 0 and (fred(hermione - 1) or  hermione == 12)
```

6) For each expression in the table below: if the expression is a valid Python expression give its *value* and its *type* (int, float, boolean, string); if not a valid expression, put "invalid" in both columns. Assume that the following variables have already been initialized as shown:

```
i = 25
s = "boogers"
x = 3.2
```

Expression	Value	Type
`i // 10`		
`i / 10`		
`i < 10`		
`pow(len(s), 2)`		
`i + x`		
`i % 10`		
`x / 2`		
`s + s`		
`25 = i`		

7) Suppose you have defined variables

```
x = 42
y = "lunchtime"
```

For each phrase below, write a Python expression that captures its meaning. Then, determine whether the expression is true or false.

x is at least 25
x is strictly between 25 and 50
x is not strictly between 25 and 50
y is either 10 or 12 characters long
x is equal to the square root of 30
it is not the case that x is positive or greater than 50
the length of y is a number that is not divisible by 3
x is an even three-digit number

8) Write a function **`next_multiple`** that takes one numeric argument x and returns the smallest number that is greater than or equal to x and is an exact multiple of 100. For example, **`next_multiple(302)`** is 400, and **`next_multiple(300)`** is 300.