

# CS5nm Midterm Exam 2 Study Guide

## Textbook Coverage

Here is an updated version of the textbook chapters, and which exams cover them.

The entries that are in *light grey italics* indicate that the material is covered, but not emphasized

Chapter	Midterm 1	Midterm 2	Final Exam
<b>1: The way of the program</b>	Entire chapter	<i>Entire chapter</i>	<i>Entire Chapter</i>
<b>2: Variables, Expressions, Statements</b>	Entire chapter	<i>Entire chapter</i>	<i>Entire Chapter</i>
<b>3: Functions</b>	3.1–3.7	<i>3.1–3.7</i> <i>3.8–end</i>	<i>Entire Chapter</i>
<b>4: TurtleWorld</b>		Entire Chapter	<i>Entire Chapter</i>
<b>5: Conditions and Recursion</b>		5.1–5.7	<i>5.1–5.7</i> <i>5.8–end</i>
<b>6: Fruitful Functions</b>		6.1–6.4	<i>6.1–6.4</i> <i>6.5–end</i>
<b>7: Iteration</b>			Entire Chapter
<b>8: Strings</b>		8.1, 8.2, 8.4	Entire Chapter
<b>9: Case Study, Word Play</b>			Entire Chapter
<b>10: Lists</b>		10.1, 10.2, 10.5, 10.6 (append and sort only)	Entire Chapter
<b>11: Dictionaries</b>			Entire Chapter
<b>12: Tuples</b>		12.1, 12.2, 12.3	Entire Chapter

## Other topics for Midterm 2

- Drawing in PyGame (ex06, ex09, [handout](#)) and TurtleWorld (Chapter 4, ex10)
- Demorgan's Law: simplifying expressions involving and, or, not
  - not (a and b) is the same as (not a) or (not b)
  - not (a or b) is the same as (not a) and (not b)
- My articles on [void vs. fruitful functions](#) and on [what print does](#)
- My article "[recursion on slices—background](#)".
  - This is a summary of material from the lecture of Monday 11/10, except for the last part on recursion.
- Lecture notes through 11/12/2008, and exercises ex01 through ex11.

## Study Guide by Chapter

### Chapter 3: Functions

- See ex13 for an example of one sample problem (we'll go over this in class on Wednesday—I won't collect
- Write the definition of a void function that takes one parameter, "msg", which is a string, and prints it twice
- Write the definition of a void function with the name `printBothTwice()` that takes two parameters, "msg1" and "msg2", concatenates them together, and then prints that twice.
  - For example, for the function call `printBothTwice("Go","Gauchos")` the output should be:

```
GoGauchos
GoGauchos
```

- Write a void function that takes two parameters with the name `printBackAndForth()` that takes two parameters, "msg1" and "msg2", and prints them twice—the first time in the order in which they were passed, and the second time in reverse order—in both cases, with a space in between the two parameters.
  - For example, for the function call `printBackAndForth("Monty","Python")`, the output should be:

```
Monty Python
Python Monty
```

### Chapter 4: TurtleWorld

If I ask you to write function definitions using TurtleWorld, I'll provide you with this brief guide on the exam itself:

- `fd(t, distance)` moves the turtle `t` forward by `distance`, and `bk(t,distance)` moves the turtle `t` backwards by `distance`
- `pu(t)` picks up the pen, and `pd(t)` puts down the pen
- `rt(t)` and `lt(t)` turn right and left by 90 degrees
- `rt(t,angle)` and `lt(t,angle)` turn the turtle right or left by the number of degrees in `angle`.
- To make a turtle named `bob`, you write:

```
world = TurtleWorld() # this is only needed once per program

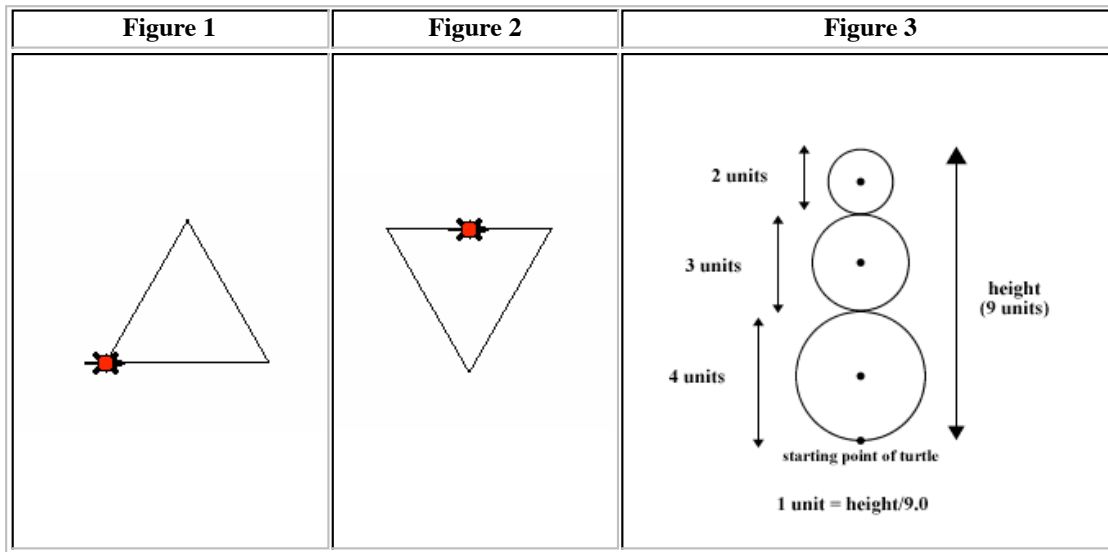
bob = Turtle() # only needed once per turtle
```

Note:

- For the exam, you do NOT need to memorize the stuff like `from TurtleWorld import *` or the `sys.append("blah blah blah")` and you don't need to put that stuff in your answers. Don't waste time writing that stuff on the exam—it will waste your time and not earn you any points!

## Practice Problems

These problems are based on the figures shown. Be sure to include comments in these function definitions where appropriate.



- Write a function `equalTri(t, length)` that takes a turtle as a parameter, and the length of one side of an equilateral triangle. Assuming that the turtle is facing "east" (towards the right hand side of the monitor when you start), draw an equilateral triangle that points up (the bottom is flat and the top points towards the top of the monitor). The triangle's bottom left point should be the place where the turtle starts and ends. After the function call the turtle should be facing the same direction that it started from, as shown in Figure 1 above:
  - Hints: interior angles of an equilateral triangle are 60 degrees—and when two angles add up to a straight line, the total is 180 degrees.
- Try the same problem, but this time call your function `deltaTri(t,length)`. The triangle should still be an equilateral triangle, but you should assume that the turtle starts in the center of a line that is assume that the triangle must face down, and that the turtle starts and ends facing east, in the center of the top line, like this:
- Assume that you have a turtle function called `drawCircle(t, r)` that takes a turtle and the radius of a circle, and draws a circle. Assume that if the turtle starts out facing east, he will finish facing east, seated at the very bottom of the circle. Using that function, create a function `drawSnowman(t, h)`, where `h` is the height of the snowman. The snowman should be structured exactly at in ex09, according to the following diagram. Be sure the turtle starts and finishes, facing east, at the bottom of the snowman (at the point indicated in the diagram below.)

## Chapter 5: Conditions and Recursion (5.1–5.7 on exam)

- Write the definition of a Python function called `isElectionYear()`
  - The function `isElectionYear()` consumes a single integer, which represents a year
  - The function returns `True` if the year is a US presidential election year, otherwise it returns `False`.
  - A year is a US presidential election year if it is the number 1789, or if the number is greater than or equal to 1792 and it is divisible by 4
- Using calls to the `check_expect()` function, write two test cases for the function `isElectionYear()`—one that should return `True`, and another that should return `False`.
- Consider this statement: "Two test cases isn't enough for `isElectionYear()`, but ten test cases is probably more than we need."
  - Why do we need more than two test cases?
  - If you were to add two more to the ones you already provided—for a total of four—which ones would you add, and why?

Fill in the blanks with what IDLE would print in the cases below.

```
>>> windy = True
>>> raining = False
>>> windy and raining
_____
>>> windy or raining
_____
>>> not windy and raining
_____
>>> windy and not raining
_____
>>> not (windy or raining)
_____
>>> not windy and not raining
_____
>>> not (windy or not raining)
_____
>>> windy and (not raining)
_____
>>>
```

Rewrite each of these expressions to remove the word `not`, without changing the meaning of the expression. The first two are done for you as an example:

before	after
<code>not (a &lt; b)</code>	<code>a &gt;= b</code>
<code>a &gt; 5 and not b == 10</code>	<code>a &gt; 5 and b != 10</code>
<code>not( (a &lt; 5) and (b == 10) )</code>	
<code>(not (a &lt; 5))and (b ==10)</code>	
<code>a &lt;=5 or not b != 10</code>	
<code>not (a==5 or b==10)</code>	

## Chapter 6: Fruitful Functions (6.1–6.4 on exam)

For these problems, you also need material from [section 8.4](#) on string slices.

- Write a fruitful function called `isTooLong()` that consumes a string, and if it is longer than 10 characters, returns `True`, otherwise it returns `False`.
  - You may use the built-in Python function `len()` which consumes a string, and returns its length as an integer
- Write a void function called `safePrint()` that consumes a string. If it is longer than 10 characters, it prints only the first

10 characters of that string, otherwise it prints the entire string.

- Write a fruitful function called `printableRegion()` that consumes a string. If it is longer than 10 characters, it returns the first 10 characters. Otherwise, it returns the entire string.

(Note: Chapter 7 is not covered on Midterm 2)

## Chapter 8: Strings (8.1, 8.2, 8.4)

Fill in the blanks from this IDLE session

```
>>> beach = "Rincon"
>>> beach[0]
_____
>>> beach[-1]
_____
>>> beach[2:4]
_____
>>> beach[3:5]
_____
>>> beach[3:3]
_____
>>> _____
```

(Note: Chapter 9 is not covered on Midterm 2)

## Chapter 10: Lists

Fill in the blanks from this IDLE session

```
>>> primes = [2,3,5,7,11,13]
>>> primes[0]
_____
>>> primes[3]
_____
>>> primes[5:]
_____
>>> primes[11:]
_____
>>> primes[1:4]
_____
>>> primes.append(13)
>>> primes[-1]
_____
>>> _____
```

(Note: Chapter 11 is not covered on Midterm 2)

## Chapter 12: Tuples

Fill in the blanks in the following IDLE session

```
>>> (x,y,z) = (1,2,3)
>>> x
_____
>>> z
_____
>>> (y,z)
_____
>>> coins = (1,2,5,10,25)
>>> coins[3:4]
_____
>>> coins[4:6]
_____
>>> coins[2:4]
_____
>>> coins[:1]
_____
>>> coins[9:]
_____
>>> coins[1:]
_____
```

```

>>> coins[4]
>>> coins[-1]
>>>

```

A combined session on strings, lists and tuples:

```

>>> x = (1,2,3)
>>> y = [1,2,3]
>>> z = '123'
>>> x[1]
>>> y[2]
>>> z[0]
>>> type(x)
>>> type(x[0])
>>> type(x[0:1])
>>> x[0:1]
>>> type(y)
>>> type(y[0])
>>> type(y[2:3])
>>> type(z[0])
>>> type(z[0:1])
>>> len(z[0:1])
>>> len(z[0:0])
>>>

```

## Drawing in TurtleWorld and PyGame

As a reminder:

- In TurtleWorld, drawing is based on moving a Turtle with functions such as
  - `fd(t,length)` and `bk(t,length)` for moving (and drawing if the pen is down)
  - `rt(t,angle)` and `lt(t,angle)` for turning
  - `pu(t)`, and `pd(t)` for picking up and putting down the pen
- In PyGame, drawing is based on  $(x,y)$  points with functions like
  - `pygame.draw.circle(screen,color,(x,y),radius,thickness)`
  - `pygame.draw.lines(screen,color,closed,pointList,thickness)`

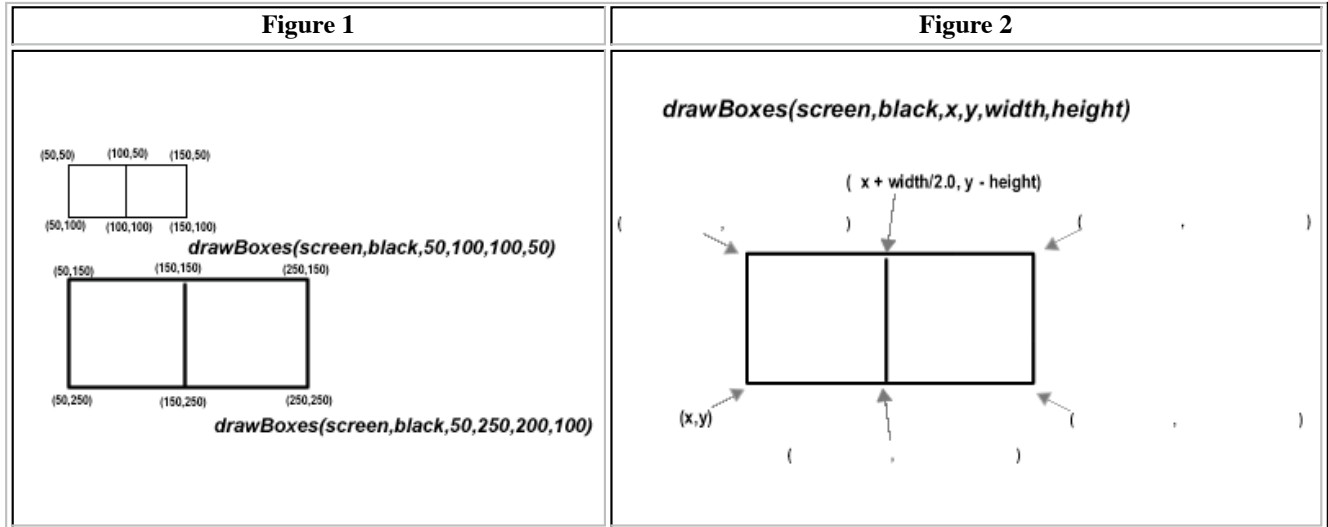
Some questions:

- When you call `pygame.draw.circle()`, `pygame.draw.lines()` or `screen.fill()`, the effects of these don't appear on the screen immediately.
  - Instead, you have to call `pygame.display.update()` first.
  - Why is PyGame built that way—i.e., why doesn't PyGame just display things immediately?
- Pygame has color and linethickness, and TurtleWorld doesn't. But ignore that difference for the moment—suppose we only use PyGame for drawing simple black and white line drawings with a line thickness of 1. There are still some differences between the two that makes some things easier in TurtleWorld, and other things easier in PyGame.
  - Focus only on the mechanics of drawing a specific figure—a skateboard, stop sign, ice cream cone, etc.
  - What kinds of figures are easier to draw in TurtleWorld than in PyGame, or vice-versa?
  - Give a specific example of a figure, and explain why it is easier to draw this in TurtleWorld than in PyGame
  - Give a specific example of a figure, and explain why it is easier to draw this in PyGame vs. TurtleWorld.
- Define the variables for values that represent the colors red, black, and white in PyGame.

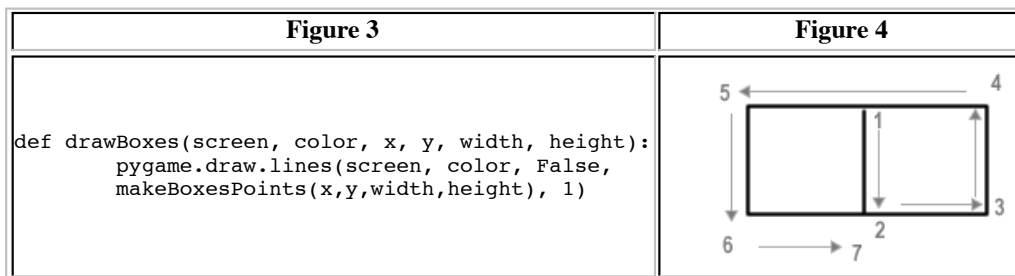
### A PyGame programming problem.

Suppose we want to make a function `drawBoxes()` to draw the following figure in PyGame.

- Figure 1 shows two specific instances of function calls, along with what they draw.
- Figure 2 shows a generic drawing. Fill in the missing points, so that they are expressed in terms of parameters to the function.



Below, Figure 3 shows a function definition for `drawBoxes()`, and a Figure 4 shows the strategy for drawing the box (i.e. the order in which the points will be drawn).



Given that, finish the incomplete function definitions shown below. Your answer should pass the test cases given.

You only need to finish the function definition for `makeBoxesPoints()`—don't worry about adding any more test cases, or any of the `import pygame` or `pygame.init()` type stuff that might be needed. Don't worry about the `while(True):` loop or the `pygame.display.update()`, or any of that stuff—assume that has already been done correctly. Just focus on this one function definition.

```
def makeBoxesPoints(x,y,width,height):
    points = []
    points.append( (x+width/2.0, y - height) ) # point 1 from diagram
    # Fill in the rest of this function...

check_expect("makeBoxesPoints 1",
    makeBoxesPoints(50,100,100,50),
    [(100,50), (100,100), (150,100), (150,50), (50,50),
     (50,100), (100,100)])
check_expect("makeBoxesPoints 2",
    makeBoxesPoints(50,250,200,100),
    [(150,150), (150,250), (250,250), (250,150), (50,150),
     (50,250), (150,250)])
```

**Answers to PyGame programming problem**

- If you want to try your solution online, download the file [drawBoxesIncomplete.py](#) and fill in your answer at the point marked # @@@
- If you want to look at a completed solution, one appears here: [drawBoxes.py](#)

(end of study guide for Midterm 2)