Review for Final Exam
Discussed in class 5/12

This review is heavy on concepts and rather light on examples. However, you have lots and lots of practice exam problems and homework problems that are also organized by chapter.

I tried to have a format that goes like this:

- Chapter summary, taken directly from textbook
- My additions to book's summary
- A list of things we did, usually I'm thinking problems, examples, or code we wrote for homework or in class work. This is an attempt to jog your memory of the types of things you can do with the material from the particular chapter.

1 Chapter 1: Computers and Programs

Summary from end of chapter:

- A computer is a universal information-processing machine. It can carry out any process that can be described in sufficient detail. A description of the sequence of steps for solving a particular problem is called an algorithm. Algorithms can be turned into software (programs) that determines what the hardware (physical machine) can and does accomplish. The process of creating software is called programming.
- Computer science is the study of what can be computed. Computer sci entists use the techniques of design, analysis, and experimentation. Com puter science is the foundation of the broader field of computing which includes areas such as networking, databases, and information manage ment systems, to name a few.
- A basic functional view of a computer system comprises a central process ing unit (CPU), main memory, secondary memory, and input and output devices. The CPU is the brain of the computer that performs

simple arith metic and logical operations. Information that the CPU acts on (data and programs) is stored in main memory (RAM). More permanent informa tion is stored on secondary memory devices such as magnetic disks, flash memory, and optical devices. Information is entered into the computer via input devices, and output devices display the results.

- Programs are written using a formal notation known as a programming language. There are many different languages, but all share the property of having a precise syntax (form) and semantics (meaning). Computer hardware understands only a very low-level language known as machine language. Programs are usually written using human-oriented, high-level languages such as Python. A high-level language must either be compiled or interpreted in order for the computer to understand it. High-level lan guages are more portable than machine language.

- Python is an interpreted language. One good way to learn about Python is to use an interactive shell for experimentation. The standard Python distribution includes a program called IDLE that provides a shell as well as facilities for editing Python programs.

- A Python program is a sequence of commands (called statements) for the Python interpreter to execute. Python includes statements to do things such as print output to the screen, get input from the user, calculate the value of a mathematical expression, and perform a sequence of statements multiple times (loop).

- A mathematical model is called chaotic if very small changes in the input lead to large changes in the results, making them seem random or un predictable. The models of many real-world phenomena exhibit chaotic behavior, which places some limits on the power of computing.

My additions:

- Zelle defines a computer as "a machine that stores and manipulates information under the control of a changeable program." There are two key elements to this definition. The first is that computers are devices for ma nipulating information. This means we can put information into a computer, and it can transform the information into new, useful

forms, and then output or display the information for our interpretation.

- "That is, we develop a step-by-step process for achieving the desired result. Computer scientists call this an algorithm.

1.1 Things we did:

This chapter was pretty broad. But we did our first print statements. We print various expressions, even though Zelle hadn't yet explained them.

We also used google Colab for all our coding, and we learned how to save, edit, and download and return to me, any Colab files that I sent to you.

2 Chapter 2: Writing Simple Programs

Summary from end of chapter:

- Writing programs requires a systematic approach to problem solving and involves the following steps: 1. Problem Analysis: Studying the problem to be solved. 2. Program Specification: Deciding exactly what the program will do. 3. Design: Writing an algorithm in pseudocode. 4. Implementation: Translating the design into a programming language. 5. Testing/Debugging: Finding and fixing errors in the program. 6. Maintenance: Keeping the program up to date with evolving needs.

- Many simple programs follow the input, process, output (IPO) pattern.

- Programs are composed of statements that are built from identifiers and ? expressions.

- Identifiers are names; they begin with an underscore or letter which can be followed by a combination of letter, digit, or underscore characters.

- Identifiers in Python are case-sensitive.

- Expressions are the fragments of a program that produce data. An expres sion can be composed of the following components: literals A literal is a representation of a specific value. For example, 3 is a literal representing the number three. variables A variable is an identifier that

stores a value. operators Operators are used to combine expressions into more complex expressions. For example, in x + 3 * y the operators + and * are used.

- The Python operators for numbers include the usual arithmetic operations of addition (+), subtraction (-), multiplication (*), division (/), and expo nentiation (**).
- The Python output statement print displays the values of a series of ex pressions to the screen.
- In Python, assignment of a value to a variable is indicated using the equal sign (=). Using assignment, programs can get input from the keyboard. Python also allows simultaneous assignment, which is useful for getting multiple input values with a single prompt.
- The eval function can be used to evaluate user input, but it is a secu rity risk and should not be used with input from unknown or untrusted sources.
- Definite loops are loops that execute a known number of times. The Python for statement is a definite loop that iterates through a sequence of values. A Python list is often used in a for loop to provide a sequence of values for the loop.
- One important use of a for statement is in implementing a counted loop, which is a loop designed specifically for the purpose of repeating some portion of the program a specific number of times. A counted loop in Python is created by using the built-in range function to produce a suitably sized sequence of numbers.

My additions:

- **print statement**

  `print(<expr>,<expr>, ...,<expr>,end="\n")`

- **simple assignment statement**

  `<variable> = <expr>`

- **input assignment statement**

  `<variable> = input(<prompt>)`

  For numbers:
  `<variable> = eval( input(<prompt>) )`

  "Beware: the eval function is very powerful and also potentially dangerous. As this example illustrates, when we evaluate user input, we are essentially allowing the user to enter a portion of our program. Python will dutifully eval uate whatever they type. Someone who knows Python could exploit this ability to enter malicious instructions. For example, the user could type an expression that captures private information or deletes files on the computer. In computer security, this is called a code injection attack, because an attacker is injecting malicious code into the running program.

  We didn't yet have ints and floats in Ch 2, but we now know enough to choose either

  `<variable> = int( input(<prompt>) )`
  or
  `<variable> = float( input(<prompt>) )`

- **simultaneous assignment statement**

  `<var1>, <var2>, ..., = <expr1>, <expr2>, ...,`
  Example:
  `sum, diff = x+y, x-y`

- **Definite Loops**

  A Python for loop has this general form:

  ```
  for <var> in <sequence>:
      <body>
  ```

  "The variable after the keyword for is called the loop index. It takes on each successive value in the sequence, and the statements in the body are executed once for each value. Often the sequence portion consists of a list of values."

- **Counted Loops**

```
for <var> in range(<expr>):
    <body>
```

2.1 Things we did:

- A bunch of "are these valid identifier" questions
- Lots of conversion-type chunks of code (like the F to C example)
- A bunch of experimenting with `range(<expr>)`
- Our first chunks of code with loops

3 Chapter 3: Computing with Numbers

Summary from end of chapter:

- The way a computer represents a particular kind of information is called a data type. The data type of an object determines what values it can have and what operations it supports.
- Python has several different data types for representing numeric values, including int and float.
- Whole numbers are generally represented using the int data type, and fractional values are represented using floats. All of the Python numeric data types support standard, built-in mathematical operations: addition (+), subtraction (-), multiplication (*), division (/), integer division (//), remainder (%), exponentiation (**), and absolute value (abs (x)).
- Python automatically converts numbers from one data type to another in certain situations. For example, in a mixed-type expression involving ints and floats, Python first converts the ints into floats and then uses float arithmetic.

- Programs may also explicitly convert one data type into another using the functions float () , int () , and round (). Type conversion functions should generally be used in place of eval for handling numeric user inputs.

- Additional mathematical functions are defined in the math library. To use these functions, a program must first import the library.

- Numerical results are often calculated by computing the sum or product of a sequence of values. The loop accumulator programming pattern is useful for this sort of calculation.

- Both ints and floats are represented on the underlying computer using a fixed-length sequence of bits. This imposes certain limits on these representations. Hardware ints must be in the range $-2^{31}...(2^{31}-1)$ on a 32-bit machine

- Python's int data type may be used to store whole numbers of arbitrary size. lnt values are automatically converted to longer representations when they become too large for the underlying hardware int. Calcula tions involving these long ints are less efficient than those that use only small ints.

My additions:

- Our first import statement was in this chapter:
  `import math`

- Table 3.2 has a list of things in math library. We definitely used: pi, sqrt(), sin() or cos(),

- A 'bit' is a binary digit: either a 0 or 1.

- You can evaluate a binary number by knowing the powers of 2, where the exponent is the "place" of the digit.

  $\cdots 2^5 = 32 \qquad 2^4 = 16 \qquad 2^3 = 8 \qquad 2^2 = 4 \qquad 2^1 = 2 \qquad 2^0 = 1$

- Decimal numbers work the same way in powers of 10, with digits 0-9.

3.1 Things we did:

- Lots of evaluating expressions with numbers, learning maybe new operators
- We converted binary numbers to decimal, and decimal to binary.
- Lots of calculations like: volume, slope, distance, length, a bit of trig
- Several problems with series and factorial-type calculations

4 Chapter 4: Objects and Graphics

Summary from end of chapter:

- An object is a computational entity that combines data and operations. Objects know stuff and can do stuff. An object's data is stored in instance variables, and its operations are called methods.
- Every object is an instance of some class. It is the class that determines what methods an object will have. An instance is created by calling a constructor method.
- An object's attributes are accessed via dot notation. Generally computa tions with objects are performed by calling on an object's methods. Acces sor methods return information about the instance variables of an object. Mutator methods change the value(s) of instance variables.
- The graphics module supplied with this book provides a number of classes that are useful for graphics programming. A GraphWin is an object that represents a window on the screen for displaying graphics. Various graph ical objects such as Point, Line, Circle, Rectangle, Oval, Polygon, and Text may be drawn in a GraphWin. Users may interact with a GraphWin by clicking the mouse or typing into an Entry box.
- An important consideration in graphical programming is the choice of an appropriate coordinate system. The graphics library provides a way of automating certain coordinate transformations.

- The situation where two variables refer to the same object is called alias ing. Aliasing can sometimes cause unexpected results. Use of the clone method in the graphics library can help prevent these situations.

My additions:

- I would definitely add more on the 'constructor' to the chapter summary. Here it is from the text: "A call to a constructor is an expression that creates a brand new object. The general form is as follows:
  `<class-name> (<param1> , <param2> , ...)`

  We often would call and assign it to a variable, like
  `circ = Circle(Point(10, 20), 5)`

- We had to work on our local computers to open graphics windows. We used IDLE and created .py files for the first time. (Can not be done in Colab.)

- We learned a new way to import a library:
  `from <library> import (<var> or <function> or *)`
  for Zelle's `graphics.py` file that becomes:
  `from graphics import *`

- Remember that `graphics.py` has to be in your working directory. (or path)

- Zelle's graphics reference materials are linked on the class web page if you need them.

4.1 Things we did:

- We did lots of problems that opened and drew objects to a graphics window on the local machine.

- We used all Zelle's shapes, and text box, modified their attributes, and drew them on the graphics window

- We moved objects

- We had different things happen when we clicked on the graphics window

5 Chapter 5: Sequences: Strings, Lists, and Files

Summary from end of chapter:

- Strings are sequences of characters. String literals can be delimited with either single or double quotes.
- Strings and lists can be manipulated with the built-in sequence opera tions for concatenation (+), repetition (*), indexing ([]), slicing ([: J), and length (len()). A for loop can be used to iterate through the charac ters of a string, items in a list, or lines of a file.
- One way of converting numeric information into string information is to use a string or a list as a lookup table.
- Lists are more general than strings.
  - Strings are always sequences of characters, whereas lists can contain values of any type.
  - Lists are mutable, which means that items in a list can be modified by assigning new values.
- Strings are represented in the computer as numeric codes. ASCII and Uni code are compatible standards that are used for specifying the correspon dence between characters and the underlying codes. Python provides the ord and chr functions for translating between Unicode codes and charac ters.
- Python string and list objects include many useful built-in methods for string and list processing.
- The process of encoding data to keep it private is called encryption. There are two different kinds of encryption systems: private key and public key.
- Program input and output often involve string processing. Python provides numerous operators for converting back and forth between numbers and strings. The string formatting method (format) is particularly useful for producing nicely formatted output.

- Text files are multi-line strings stored in secondary memory. A text file may be opened for reading or writing. When opened for writing, the existing contents of the file are erased. Python provides three file-reading meth ods: read(), readline(), and readlines(). It is also possible to iterate through the lines of a file with a for loop. Data is written to a file using the print function. When processing is finished, a file should be closed.

My additions:

- We used the split method often enough that I'll add it here:
  `<str>.split(<separator>)`
  `<separator>` : The string splits at this specified separator. If is not provided then any white space is a separator.
  it returns a list of strings that results from breaking at the separator.
- We also learned the `<str>.append` method in this chapter.
- There is a list of common string methods in table 5.2
- Reading and writing files also must be done on the local machine–using IDLE and .py files.

5.1 Things we did:

- We did a couple problems of using strings to convert scores to letter grades
- The is the chapter we did a little bit of encoding letters to numbers and then turning the numbers back into messages
- We did lots of operations on strings and lists
- We counted words, sliced strings, indexed strings, split strings and used the pieces
- We added things to lists, replaced things in lists....

6 Chapter 6: Defining functions

Summary from end of chapter:

- A function is a kind of subprogram. Programmers use functions to reduce code duplication and to help structure or modularize programs. Once a function is defined, it may be called multiple times from many different places in a program. Parameters allow functions to have changeable parts. The parameters appearing in the function definition are called formal pa rameters, and the expressions appearing in a function call are known as actual parameters.

- A call to a function initiates a four-step process:

  1. The calling program is suspended.
  2. The values of actual parameters are assigned to the formal parame ters.
  3. The body of the function is executed.
  4. Control returns immediately following the function call in the calling program. The value returned by the function is used as the expression

- The scope of a variable is the area of the program where it may be refer enced. Formal parameters and other variables inside function definitions are local to the function. Local variables are distinct from variables of the same name that may be used elsewhere in the program.

- Functions can communicate information back to the caller through return values. In Python, functions may return multiple values. Value-returning functions should generally be called from inside an expression. Functions that don't explicitly return a value return the special object None.

- Python passes parameters by value. If the value being passed is a mutable object, then changes made to the object may be visible to the caller.

My additions:

- Sorting out the language associated with functions: I'll separate defining a function, calling a function, passing parameters to a function, and a function returning values.
- Defining a function:

```
def <name>(<formal-param1>, <formal-param2>, ...  ):
    <body>
```

- Calling a function: This is usually from the main part of the code (which might be the main function, or just the top level of code):

```
<functionName>(<actual-param1>, <actual-param2> ...)
```

- Passing parameters to a function:
  In the definition above, there are `<formal parameters>`, but when I call the function, i used the terms `<actual parameters>`. The `<formal parameters>` are the names used in the body of the function definition. The `<actual parameters>` are the names used wherever the function is called from.
- The function returning a value:
  In the definition of the function, in the ¡body¿, there could appear a `return` statement.

```
return <var1>, <var2>, ...
```

  If there is no return statement, the function will return `None`.

6.1 Things we did:

- We converted many old problems to functions. Some were calculations, some did things to strings....
- We wrote lots of functions that did area, volume or perimeter type calculations
- We wrote functions to do things to graphics objects

7 Chapter 7: Decision Structures

Summary from end of chapter:

- Decision structures are control structures that allow a program to execute different sequences of instructions for different cases.
- Decisions are implemented in Python with if statements. Simple decisions are implemented with a plain if. Two-way decisions generally use an `if - else`. Multi-way decisions are implemented with `if - elif - else`.
- Decisions are based on the evaluation of conditions, which are simple Boolean expressions. A Boolean expression is either true or false. Python has a dedicated bool data type with literals `True` and `False`. Conditions are formed using the relational operators: `<, <=, !=, ==, >, and >=`.
- (JCK we didn't talk about this one. Feel free to skip it.) Some programming languages provide exception handling mechanisms which help to make programs more "bulletproof." Python provides a try-except statement for exception handling.
- Algorithms that incorporate decisions can become quite complicated as decision structures are nested. Usually a number of solutions are possible, and careful thought should be given to produce a correct, efficient, and understandable program.

My additions:

- I introduced the `time` library:
  `import time`
- From that, we used
  `time.sleep(<number>)`
  where the `number` is the number of seconds the computer waits before doing the next thing.

7.1 Things we did:

- We converted several old problems to if/elif/else format. I think we did at least one of the number to letter grade problems.
- We checked to see if numbers were positive or negative to prevent taking the square root of a negative number.
- We did several "if this, print that" type of programs.
- We wrote our first animation! It was the program to animate a circle bouncing around the graphics window. We could do the animation as soon as we had the for loops, but we needed to if to keep the circle from just going out of the graphics window.

8 Chapter 8: Loop Structures and Booleans

Summary from end of chapter:

- A Python for loop is a definite loop that iterates through a sequence.
- A Python while statement is an example of an indefinite loop. It continues to iterate as long as the loop condition remains true. When using an indef inite loop, programmers must guard against the possibility of accidentally writing an infinite loop.
- One important use for an indefinite loop is for implementing the program ming pattern interactive loop. An interactive loop allows portions of a program to be repeated according to the wishes of the user.
- A sentinel loop is a loop that handles input until a special value (the sentinel) is encountered. Sentinel loops are a common programming pattern. In writing a sentinel loop, a programmer must be careful that the sentinel is not processed.
- Loops are useful for reading files. Python treats a file as a sequence of lines, so it is particularly easy to process a file line by line using a for loop. In other languages, a file loop is generally implemented using a sentinel loop pattern.

- Loops, like other control structures, can be nested. When designing nested loop algorithms, it is best to consider the loops one at a time.
- Complex Boolean expressions can be built from simple conditions using the Boolean operators `and, or, and not`. Boolean operators obey the rules of Boolean algebra. DeMorgan's laws describe how to negate Boolean expressions involving and and or.
- Nonstandard loop structures such as a loop and a half can be built using a while loop having a loop condition of True and using a break statement to provide a loop exit.
- Python Boolean operators and and or employ short-circuit evaluation. They also have operational definitions that allow them to be used in cer tain decision contexts. Even though Python has a built-in bool data type, other data types (e.g., int) may also be used where Boolean expressions are expected.
- GUI programs are generally event driven and implement carefully de signed event loops to control user interaction. Interactions are called non modal when the user is in control of what happens next and modal when the application dictates what the user must do next.

My additions:

- The while loop has the form:

  ```
  while <condition>:
        <body>
  ```

  "The body of the loop executes repeatedly as long as the condition remains true. When the condition is false, the loop terminates."
- Truth tables are when you write out all the possible values of a Boolean expression
- We showed that you can duplicate a lot of Boolean algebra if you let True = 1 and False = 0.

- "Python follows a standard convention that the order of precedence from high to low is not, followed by and, followed by or. So the expression would be equivalent to this parenthesized version:
  `(a or ((not b) and c))`
  I'm generally not a fan of parentheses, but here they are meant to explain.

8.1 Things we did:

- We did lots of series-type problem (like adding or multiplying terms in a series.)
- We did a couple problems involving finding primes.
- I've asked a couple truth table questions
- I've had you convert for loops to while loops

9 Chapter 9: Simulation and Design

Summary from end of chapter:

- Computer simulation is a powerful technique for answering questions about real-world processes. Simulation techniques that rely on probabilistic or chance events are known as Monte Carlo simulations. Computers use pseudo-random numbers to perform Monte Carlo simulations.
- Top-down design is a technique for designing complex programs. The basic steps are:
  1. Express an algorithm in terms of smaller problems.
  2. Develop an interface for each of the smaller problems.
  3. Express the algorithm in terms of its interfaces with the smaller prob lems.
  4. Repeat the process for each of the smaller problems.
  5. Top-down design was illustrated by the development of a program to simulate the game of racquetball.

- Unit-testing is the process of trying out each component of a larger pro gram independently. Unit-testing and bottom-up implementation are use ful in coding complex programs.
- Spiral development is the process of first creating a simple version (prototype) of a complex program and gradually adding features. Prototyping and spiral development are often useful in conjunction with top-down de- sign.
- Design is a combination of art and science. Practice is the best way to become a better designer.

My additions:

- I didn't talk much about the Design parts of this chapter. I won't test them directly. Reading those sections may help you with the more complex design, but I find it is largely a matter of personal preference which works best for each person.
- We learned about the `random` library
  `import random`
  From that library, we used:
  - The `random()` function:
    `random()` returns a number from [0, 1). (Includes 0 but not 1.)
  - The `randrange()` function takes three possible parameters:
    `randrange(<startInt>, <endInt>, <step>)`
    and returns and integer value starting at `<startInt>`, ending at `(<endInt> - 1)`, going up by `<step>`.
- I also introduced `matplotlib` and `pyplot`. You will not be asked to code any plots from those libraries. I might include some code in the Colab file that you could use if you like.
- We used `plot` and `hist` from those libraries–but again, they are not required for the exam.

9.1 Things we did:

- We did several dice rolling problems

- I simulated C14 decay in a Colab file
- I simulated roulette.
- A homework problem or two did Blackjack simulations.
- One problem estimated the value of π by running a simulation

10 Chapter 10: Defining Classes

Summary from end of chapter:

- An object comprises a collection of related data and a set of operations to manipulate that data. Data is stored in instance variables and manipulated via methods.
- Every object is an instance of some class. It is the class definition that determines what the attributes of the object will be. Programmers can create new kinds of objects by writing suitable class definitions.
- A Python class definition is a collection of function definitions. These func tions implement the methods of the class. Every method definition has a special first parameter called self. The actual parameter of self is the object to which the method is being applied. The self parameter is used to access the attributes of the object via dot notation.
- The special method `__init__` is the constructor for a class. Its job is to initialize the instance variables of an object.
- Defining new objects (via class) can simplify the structure of a program by allowing a single variable to store a constellation of related data. Objects are useful for modeling real-world entities. These entities may have complex behavior that is captured in method algorithms (e.g., a projec tile), or they may be little more than a collection of relevant information about some individual (e.g., a student record) .
- Correctly designed classes provide encapsulation. The internal details of an object are hidden inside the class definition so that other portions of the program do not need to know how an object is implemented. This separation of concerns is a programming convention in Python;

the in stance variables of an object should only be accessed or modified through the interface methods of the class.

- Most GUI systems are built using an object-oriented approach. We can build novel GUI widgets by defining suitable classes. GUI widgets can be used to construct custom dialogs for user interaction.

My additions:

- Defining a class:

```
class <class-name>:
      <method-definitions>
```

  - The first method is almost always the constructor or init method:

```
def __init__ (self, <param1>, <param2>, ...  ):
    <body>
```

  - The first parameter of every method in the definition is (usually) called `self`, and it refers to the specific object on which the method is acting..

- We programmed our first widgets! A graphical widget (also graphical control element or control) in a graphical user interface is an element of interaction, such as a button or a scroll bar.

- I added all Zelle's code from chapter 10 to our shared google drive (and it's linked from class page.)

10.1 Things we did:

- We worked through the book's MSDie class

- We modified classes that Zelle had written for us.