CPSC 230 Final Exam Review

This is a list of broad topics you should be familiar with for the final exam. It is not a comprehensive list, but if you understand these topics and can answer the example questions, you should do well on the exam. I suggest you also review your classworks, programming assignments, class notes, lecture slides, and zyBooks for more information. As always, reach out with questions!

Computer Science Basics

- Von Neumann Architecture
- Binary number system
  - Base 10 to binary conversions
  - Binary to base 10 conversions
- Files vs. directories
- Basic computer components and their functions (not the programming kind of function)
- How to execute a Python script using the terminal

  Example Questions:
  1. Describe the basic components of the von Neumann Architecture and name common I/O devices.
  2. Convert 1101101_2 to base 10
  3. Convert 73 to base 2
  4. What is the purpose of volatile main memory (e.g. RAM) in a computing system?
  5. When I execute a Python script, which hardware component performs the arithmetic operations?

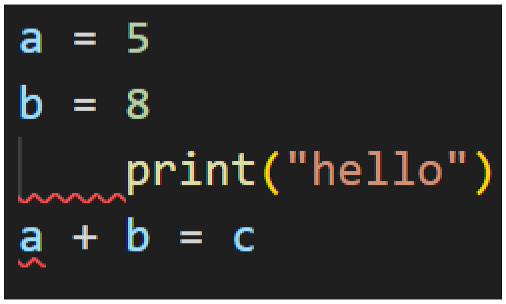
Python Basics

- Basic data types
- Built-in functions: print(), input(), type(), isinstance(), len()
- Constructors for each data type (type casting)
- Variables
  - Assignment
  - Naming rules

    - Naming conventions
        - camelCase vs. snake_case
- Comments and why we use them
- Whitespace and where it matters
- Arithmetic Operations and Precedence

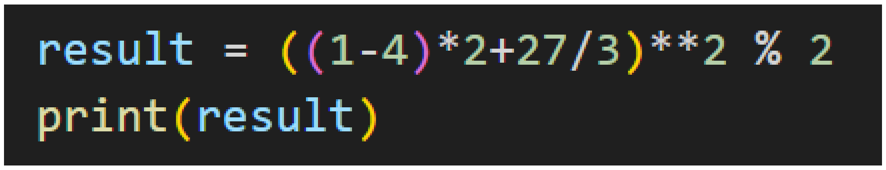
Example Questions:

1. How do I assign the value 10 to the variable x?
2. Which basic data type allows for us to store decimal numbers without removing information?
3. How could I ask the user to enter some text and then print that text to the terminal?
4. How many argument are passed to the isinstance() function? The type() function? What do they do?
5. Is it important to maintain a consistent variable naming style throughout a script?
6. Why do I insist that you comment your code? (Hint: it is not because I am a sadist)
7. What is wrong with the following lines of code?



```
a = 5
b = 8
    print("hello")
a + b = c
```

8. What is the output of the following statement?



```
result = ((1-4)*2+27/3)**2 % 2
print(result)
```

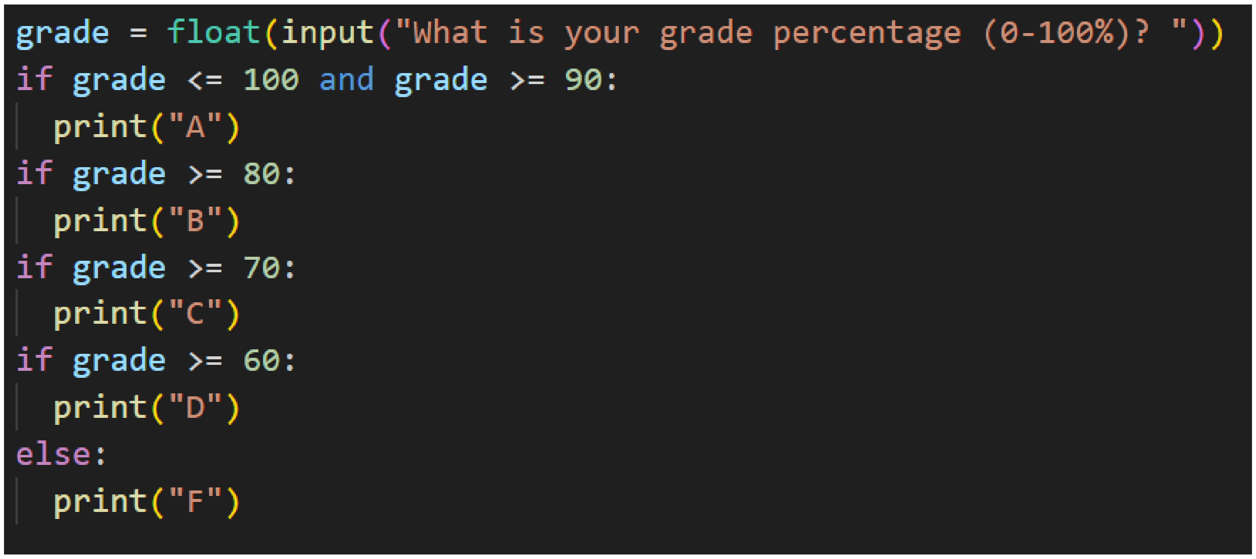
Conditionals

- Branching logic and Boolean questions
- Syntax
- Boolean operators
    - == vs. =

- Compound Boolean expressions
- if vs. if-else vs. if-elif-else

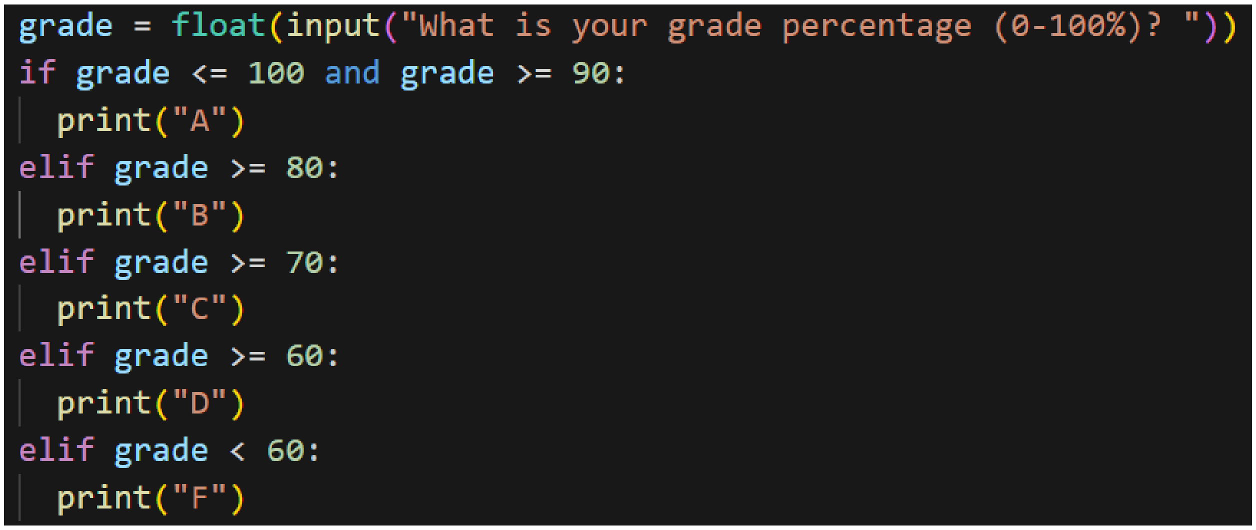
Example Questions:

1. What must be the result of a conditional statement in Python? Why is this?
2. What is the difference between == and =?
3. If I want to print someone's letter grade, which structure will work better?



```python
grade = float(input("What is your grade percentage (0-100%)? "))
if grade <= 100 and grade >= 90:
    print("A")
if grade >= 80:
    print("B")
if grade >= 70:
    print("C")
if grade >= 60:
    print("D")
else:
    print("F")
```

OR



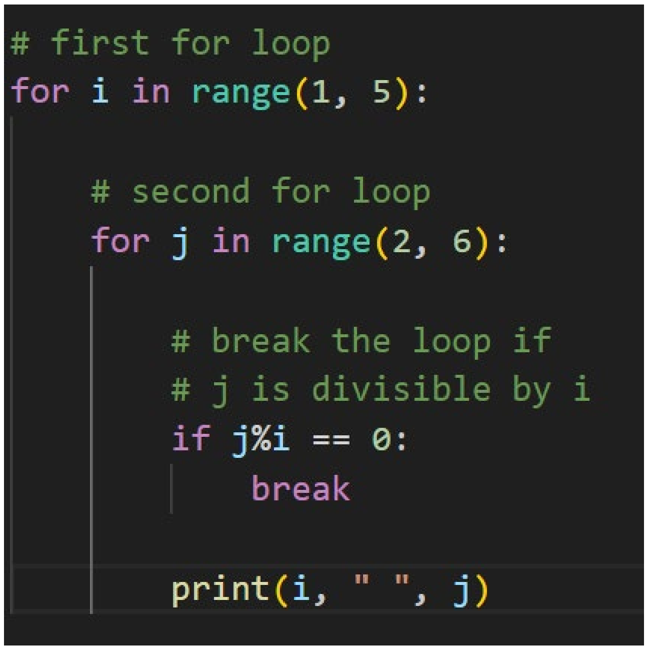
```python
grade = float(input("What is your grade percentage (0-100%)? "))
if grade <= 100 and grade >= 90:
    print("A")
elif grade >= 80:
    print("B")
elif grade >= 70:
    print("C")
elif grade >= 60:
    print("D")
elif grade < 60:
    print("F")
```

Loops

- *while* loops
    - syntax
    - logic
    - infinite loops
    - break and continue
- *for* loops
    - syntax
    - logic
    - when to use them instead of while loops
    - functions: range(), len()
    - iterating over different iterables
        - range()
        - strings
        - collections: lists and dictionaries
    - break and continue

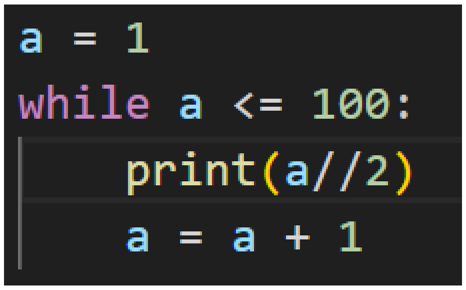
Example Questions:

1. What might cause a *while* loop to execute indefinitely (an infinite loop)? With the existing modules and concepts we have talked about, do you think it is possible for a *for* loop to execute indefinitely?
2. What is the output of the following program?



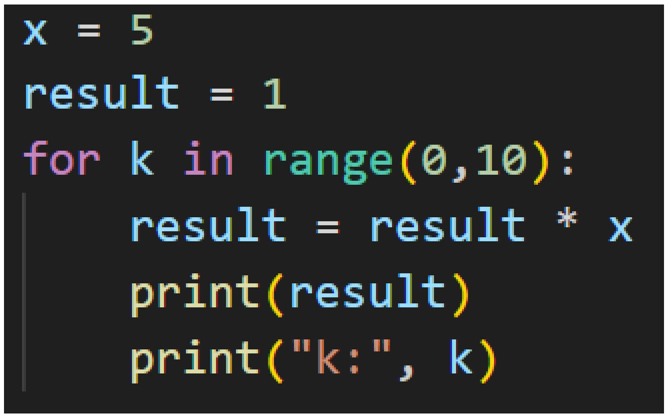
```python
# first for loop
for i in range(1, 5):

    # second for loop
    for j in range(2, 6):

        # break the loop if
        # j is divisible by i
        if j%i == 0:
            break

        print(i, " ", j)
```

3. What is the output of the following program?



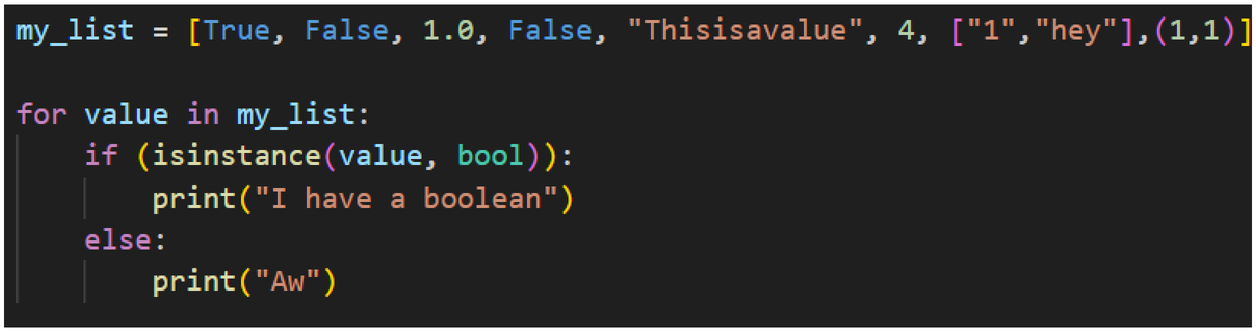
```python
a = 1
while a <= 100:
    print(a//2)
    a = a + 1
```

4. Using concepts that we have talked about, can you think of a way to iterate over the digits in an integer and take the sum? For example, the result of your short program would be 20 if your number was 7445
5. How many times will the for loop execute in the following code snippet?



```python
x = 5
result = 1
for k in range(0,10):
    result = result * x
    print(result)
    print("k:", k)
```

6. How many times will "I have a Boolean" be printed in the following code snippet?



```python
my_list = [True, False, 1.0, False, "Thisisavalue", 4, ["1","hey"],(1,1)]

for value in my_list:
    if (isinstance(value, bool)):
        print("I have a boolean")
    else:
        print("Aw")
```

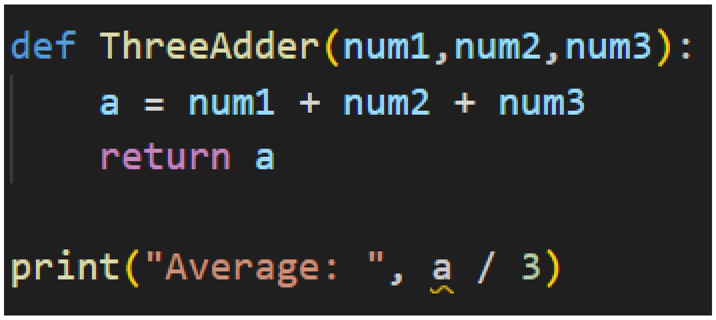
Functions

- Namespaces/Scopes
- Syntax
  - Defining a function
  - Invoking a function
- Use cases

- *return* statement
- Argument passing
    - Named vs ordered
- Default parameters
- Nested function calls

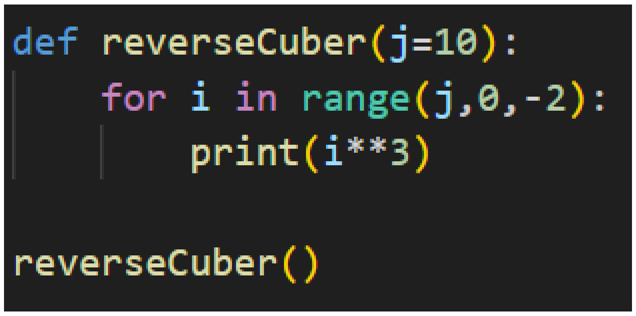
Example Questions:

1. What is the difference between a built-in function and a user-defined function?
2. Why can *a* not be used outside of the following function?



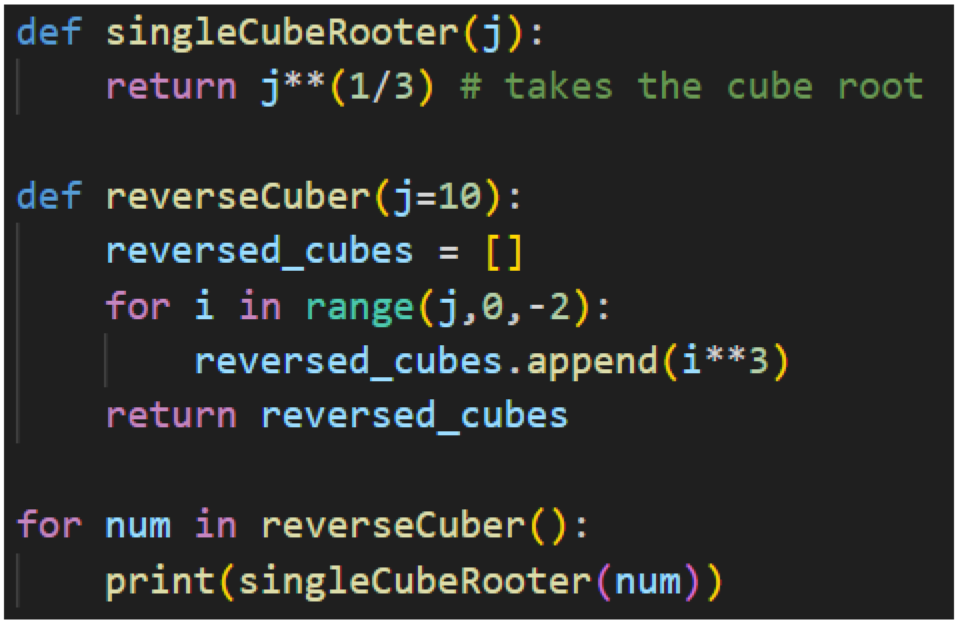
```python
def ThreeAdder(num1,num2,num3):
    a = num1 + num2 + num3
    return a

print("Average: ", a / 3)
```

3. What is the default returned value by a function if the programmer omits a *return* statement?
4. Can you pass arguments to a function in a different order than the parameters? If so, how?
5. How can you set a default argument in the function definition?
6. What do you expect the output from this program to be?



```python
def reverseCuber(j=10):
    for i in range(j,0,-2):
        print(i**3)

reverseCuber()
```

7. If I slightly modify reverseCuber(), define another function and use them like below, what is the output?



```python
def singleCubeRooter(j):
    return j**(1/3) # takes the cube root

def reverseCuber(j=10):
    reversed_cubes = []
    for i in range(j,0,-2):
        reversed_cubes.append(i**3)
    return reversed_cubes

for num in reverseCuber():
    print(singleCubeRooter(num))
```

Strings

- Mutable or immutable?
- Properties
- Methods: .split(), .replace(), .upper(), .lower(), .islower(), .isalpha(), etc.
- Using strings with *for* loops
- Concatenation

Example Questions:

1. What makes strings different from the other basic data types?
2. What is the returned value of .isalpha()?
3. Construct a function named isupper_v1() which takes in a string as an argument and returns whether a string has all lowercase letters **without using .isupper()**
4. How can you combine two or more strings into a new string variable?

Collections

- 4 types: lists, tuples, sets, dictionaries
- Importance
- Lists vs. tuples
- List methods (e.g. .append(), .insert(), .remove(), .pop(), etc)
- List and tuple indexing and slicing
- Use case for tuples
- Unique properties of sets
- Set operations/methods (e.g. .intersection(), .union(), .difference(), .symmetric_difference())
- Dictionaries
    - Information storage format
    - Accessing keys
    - Methods for using as an iterable in a *for* loop
    - Methods (e.g. .update(), .get(), .values(), .items(), .keys(), etc.)

Example Questions:

1. What makes collections different from base data types? Would you consider strings to be a collections? Why or why not?
2. If you were to store the coordinates for fixed points in the cartesian plane, which collection would you use and why?
3. What is the difference between .remove() and .pop()?
4. If I have two sets of numbers, which set method would return the items that are only present in one set but not in the other?
5. Write a function key_val_swap() which takes in a dictionary as an argument and returns a dictionary with each key-value pair swapped (i.e. keys become values and values become keys)
6. Why might a programmer use the .keys() dictionary method?
7. What are the two ways to access the value associated with a particular key in a dictionary?

<u>File Processing</u>

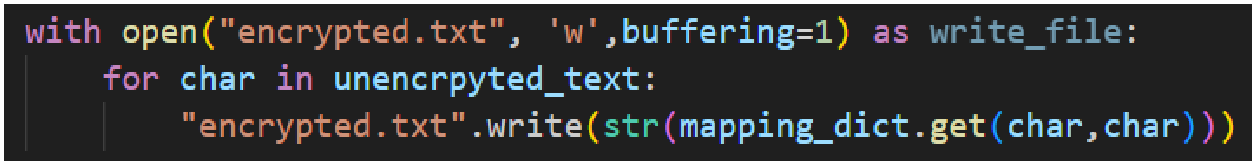
- What is a file?
- Opening and closing files
    - Opening modes
- Reading and writing from/to file objects
    - Read methods: .read(), .readlines()

- Write method: .write()

- Absolute vs relative pathing
- os.path.join() function and it’s benefits
- Context managers (*with* statement)

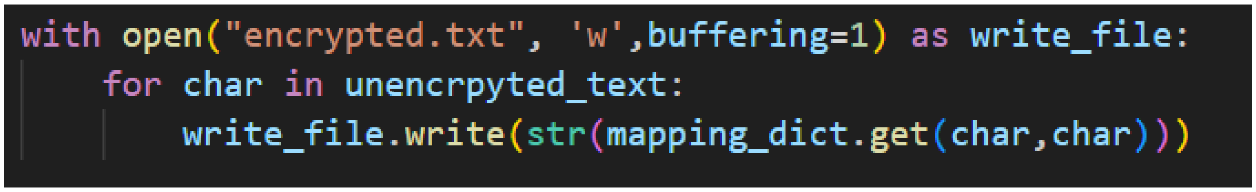
Example Questions:

1. Is open() a function or a method?
2. Which of the following is correct implementation of .write()?



```python
with open("encrypted.txt", 'w',buffering=1) as write_file:
    for char in unencrpyted_text:
        "encrypted.txt".write(str(mapping_dict.get(char,char)))
```

OR



```python
with open("encrypted.txt", 'w',buffering=1) as write_file:
    for char in unencrpyted_text:
        write_file.write(str(mapping_dict.get(char,char)))
```

3. What is returned by .read()? What about .readlines()?
4. Assume you are reading a poem file into your Python module which has no whitespace other than newline characters. If you want to modify every 3rd line in the poem and write the updated poem back to the same file, would you choose to use .read() or .readlines()? Why? Also, what file mode would you choose when opening the original file for modification?
5. What is the main benefit of using a context manager in your file processing workflow?

Exception Handling

- Purpose
- Keywords
- Syntax
- When it might be useful and when it might be undesirable

Example Questions:

1. What two basic keywords allow you handle errors without halting your program to display the error message?
    - Why might it be beneficial to prevent program halting? Why might it be unwise to halt your program?
2. Can you handle all possible errors without specifying each error individually after the *except* keyword? In other words, is there a “catch-all” mechanism native to Python?