- Have paper
- Download psets/instructions and these notes
- Turn on do not disturb

- COMMON MISTAKES TO CHECK FOR:
    - Make sure to use appropriate '=' or '==' - usually == in if statement, = is for assignment
    - Make sure to put ':' in function
    - print(foo, bar) → 25 12 (not 25, 12) - if it asks for precise outcome type EXACT outcome
    - Make sure to always have correct ""
    - Capitalization
    - Commas in correct place after ""
    - Call function
    - n +1 in range
    - Use elif
    - For i in range (x)
    - Close parenthesis
    - IF IN YOUR FOR LOOP THERE IS (i + 1) OR IT IS LOOKING AHEAD, YOUR RANGE NEEDS TO BE -1 SO ITS NOT OUT OF RANGE
    - Don't forget else statement!!
    - CALL FUNCTION
    - CLOSE PARENTHESIS AND QUOTES

UNIT 1

- Practice binary conversion - 10101= 1 * 16 + 1 * 4 + 1 * 1 = 21
- Divide $(71)_{10}$ successively by 2 until the quotient is 0:
- 71/2 = 35, remainder is 1
  35/2 = 17, remainder is 1
  17/2 = 8, remainder is 1
  8/2 = 4, remainder is 0
  4/2 = 2, remainder is 02/2 = 1, remainder is 0
  1/2 = 0, remainder is 1
- 93 1011101
- 46 1
- 23 0
- 11 1
- 5 1
- 2 1
- 1 0
- 0 1
- LINUX: operating system; updated from UNIX
- Shortcuts in py
  - ls: shows you all files
  - mv: rename
  - rm: delete file
  - mkdir: make directory/new folder
    - mkdir name of folder
  - rmdir: remove directory/folder
- Python became popular bc:
  - Its dynamically typed (not strongly typed)
  - Simple syntax
  - Pythonic one-liners (can accomplish a lot in one line)
  - English-like commands
  - Intuitive data structure (lists, tuples, sets, dictionaries)
  - Large amounts of prewritten software libraries (modules)
- Object oriented programming:
  - OOP can be summarized by 6 key concepts:
    - Objects that provide encapsulation (encapsulation is a way of combining data and beh)
    - Classes that implement inheritance within class hierarchies
    - Messages that support polymorphism across objects
  - Some OOP languages
    - Simula

        - Smalltalk
        - C++
        - Others (java, eiffel)
- Print (‘I miss scratch’, end=’’)
- Variable
    - Foobar = 17
    - Type (foobar)
    - Int
    - Foobar = 3.14
    - type(foobar)
    - Float
    - foobar=hi
    - Foobar+’ blah’
    - Hi blah (concatenating)
- Can have _ before
- Assignment statement
    - You can assign the value of an expression to a variable
        - Variable = expression
    - Arithmetic expressions
        - +
        - -
        - *
        - ** - exponent
        - /
        - // - answer is truncated (rounds down to whole num - after decimal deleted)
        - % - computes remainder
- Operator precedence
    - PEMDAS
    - (**,*,/,//,%,+,-)
- +=,-=
    - X=5
    - X+=2
    - 7
- Capitalize variables whose value is constant
- For in container
- cd: change directory
    - ls: what’s in this folder?
    - Code new_file.txt
- * can repeat string integer number of times
    - “row” * 3
    - row row row
- Boolean-valued expressions
    - < : less than
    - == : value is the same bw two numbers

  - > : greater than
  - <= : less than or equal to
  - >= : greater than or equal to
  - != : not equal to
- Conditional statements
  - If boolean expression:
    Python statement
  - If boolean expression:
    Python statement
  - Else:
    Python statement 2
  - Elif : abbreviation of else if
- F string
  - {} placeholders
- Floating point imprecision
- Memory leak - when you need a dictionary for a program, you have to recycle it. If you never release it, in a loop, it keeps asking for more mem, and the program has to look around to allocate memory from other places, causing it to slow down
- <u>Demorgan's Law</u>: not (a and b) equals (not a) and (not b); (a and b) is False only if either a is False or b is False.
- Or - used in python always inclusive or
- When you distribute not everything changes, even operator
- Not (p and q) → not p or not q
- Every not has to be distributed
- Fibonacci sequence - 1 1 2 3 5 8

- Cant add a string and an int
  - Which of the following code segments will print the message **Catch 22**?
    - I.   print("Catch", 11 + 11)
    - II. print("Catch " + (11 + 11))
    - III. print(f"Catch {22}")
    - Only I and III
- def main():
-     print("She said, \"", end="")
-     print("Never put off till tomorrow... ", end="")
-     print("\nWhat you can do")
-     print("The day", end="")
-     print()
-     print("After tomorrow!\"")
- She said, "Never put off till tomorrow... What you can do The day After tomorrow!"

```
-  Precisely how many values do each of the following loops print?
   (In other words, how many times does each loop iterate?)
      -  for j in range(3000):
      -      print(j)
      -  3000
      -  for j in range(-5, 5):
      -      print(j)
      -  10
      -  for j in range(50, -10, -8):
      -      print(j)
      -  8
-  def main():
-     concentration = "fred"
-     fred = "computer science"
-     computer = "department"
-     department = "student"
-     student = "concentration"
-
-     sentence(student, "fred", concentration)
-     sentence(fred, computer, department)
-
-  def sentence(concentration, fred, foo):
-     print("Many a " + foo + " in the "
-                    + fred + " of " + concentration)
-  The following will be printed: Many a fred in the fred of
   concentration Many a student in the department of computer
   science And the values of the parameters of sentence are: foo =
   fred, fred = fred, concentration = concentration foo = student,
   fred = department, concentration = computer science
```

UNIT 2

- **s = "Your Friend Is Stuck. You Give Me Money, I Make Him Unstuck."**
- .count("stuck") only counts "stuck" not "unstuck"
- .find returns a number indicating the position of the first letter of the word including spaces
- s.find('Friend')   - 5 bc first letter is in 5th position
- s[0:5].upper() - YOUR
- s.find('enemy', 4) - if its a word that is not in the sentence it returns -1
- s[2].islower() - boolean - returns true of false if it is lower/uppercase
- x = [5, 3, 5, 7, 5, "banana"]
    - x.index(5) is 0 bc it returns just the first instance
    - X[0:2] returns just [5, 3]
- **y = ['a', 'b', 'c']**
    - y[2] = 'd'
    - Y = ['a', 'b', 'd'] - replaces
- **y = ['a', 'b', 'c']**
    - y.insert(2, 'a')
    - Y = ['a', 'b', 'a', 'c'
- Set every element of **foobar** to the value **5.**

```
- foobar = [1, 2, 3, 4]
- for i in range(len(foobar)):
- foobar[i] = 5
-
- print(foobar)
```

- Swap the first value in the **foobar** list with the last value in the list.

```
- foobar = [1, 4, 9, 16, 25]
- foobar.insert(0, foobar[-1])
- foobar.pop(-1)
-
- print(foobar)
```

- ''.join(list_name) will turn list into string (make sure if there are numbers in list, you convert them to str first)

```
- for i in words:
- alph_word = sorted(i)
```

- Sort function -

```
- def alphabetical(words):
- alphabetical_list = []
```

```
-
-   for i in words:
-   alph_word = ''.join(sorted(i, key=str.lower))
-   alphabetical_list.append(alph_word)
-   return alphabetical_list
```

- sorted('rat') - a, r, t
- Sorted ('raT') - T, a, r - capital sort before lower
- [1, 2, 3] + [1, 1, 1] → [1,2,3,1,1,1]
- [1, 2, 3] - list
- (1, 2, 3) - tuple
- Each character corresponds with a number (Unicode)
- ord() - tells you the unicode number is
- chr() - tells you the letter fir a unicode
- chr(random.randint(ord('a'), ord('z'))) - picks a random char
- random.choice('aeio')
- IF IN YOUR FOR LOOP THERE IS (i + 1) OR IT IS LOOKING AHEAD, YOUR RANGE NEEDS TO BE -1 SO ITS NOT OUT OF RANGE
- Difference bw lists and tuples is that tuples are immutable (can't change them)
- Strings are like tuples of characters
- Lists are mutable!
- Append, pop, sort (not sorted)
- Import argv
- BIG O NOTATION
    - O(1) - pigeon analogy; takes the same amnt of time no matter how much data
    - O(N) - changes linearly depending on amnt of data; two times the data takes twice amnt of time; often single loop
    - O(N^2) - if n doubles, time changes quadrtatically; often nested loops
- Sys. before argv[]

RECURSIVE:
```
def mystery(n):
  if n <= 0:
    return 0
  return mystery(n // 2) + 1
```

Answer = 5

```
def recurse(a_list):
  if a_list == []:
    return 0
  else:
    return 1 + recurse(a_list[1:])
```

```
recurse([4, 9, 'blah', 3.21])
```

Answer = 4

```
def power(x, n):    # raise x to the n'th power
  if n == 0:
    return 1.0
  elif n > 0:
    return x * power(x, n-1)
  else:
    return 1.0 / power(x, -n)
```

Take advantage of fact that if n is even you can do (x^n/2)^2 to make it more efficient

Answer =

```
def power(x, n):
	if n == 0:
		return 1.0
	elif n > 0:
		if (n % 2 == 0):
			new = power(x, n//2)
			return new * new
		else:
			return x * power(x, n-1)
	else:
		return 1.0 / power(x, -n)
```

Total calls = 12

Prints sum of command line args:

```
import sys

total = sum(int(argument) for argument in sys.argv[1:])
print(f"The sum of the args is: {total}")
```

Write a short *Python* program that takes 2 command-line arguments: the name of an output file name and an integer **n**. Your program should use a loop to *write* **n** random integers into the file, one per line. Each random integer should be a value between 0 and 36 (inclusive).

```
import random
import sys

def roulette():
output = open(sys.argv[1], "w")
n = int(sys.argv[2])

if len(sys.argv) != 3:
print("Error. Correct usage: roulette.py output_file_name n")
else:
for x in range (n):
random_int = str(random.randint(0, 37))
output.write((random_int) + "\n")

output.close()

roulette()
```

Palindorome:

```
def is_palindrome(a_string):
b_string = a_string.lower()
if len(b_string) <= 1:
return True
else:
if (b_string[0]).lower() == (b_string[-1]).lower():
return is_palindrome(b_string[1:-1])
else:
return False
print(is_palindrome ("kayaK"))

def mystery(x):
      print(x % 10,end="")
      if x // 10 != 0:
            mystery(x//10)
      print(x % 10,end="")
```

Which of the following is printed as a result of the method call mystery (1234)
43211234

BIG O:

Big O notation is used to describe the time complexity (how the runtime grows with input size) of an algorithm or function. It gives us an idea of how efficient the algorithm is in terms of its time complexity.

A. O(log n):

- Logarithmic time complexity.
- An algorithm with O(log n) time complexity reduces the problem size by a constant factor with each step.
- Commonly seen in binary search algorithms.
- As the input size grows, the runtime increases at a slow rate.
- Ex: Binary search

```
def binary_search(arr, target):
    low, high = 0, len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

B. O(n):

- Linear time complexity.
- An algorithm with O(n) time complexity has a linear relationship between the input size and the runtime.
- The runtime grows linearly as the input size increases.
- Example: Iterating through an array or list.
- Linear search:

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1

For k in range (3, n, 2);
    print(k)
```

C. O(n log n):

- Linearithmic time complexity.
- Often seen in efficient sorting algorithms like Merge Sort and Quick Sort.
- The runtime grows faster than linear but slower than quadratic (quadratic is O(n^2)).
- Example: Merge Sort takes O(n log n) time to sort a list.

D. O(n^2):

- Quadratic time complexity.
- An algorithm with O(n^2) time complexity has a quadratic relationship between the input size and the runtime.
- As the input size increases, the runtime grows quadratically.
- Example: Nested loops iterating through an array.
- Bubble sort:

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

E. O(n!):

- Factorial time complexity.
- An algorithm with O(n!) time complexity grows exponentially with the input size.
- It is highly inefficient and should be avoided for large input sizes.
- Example: Generating all possible permutations of a set.

```
from itertools import permutations

def generate_permutations(elements):
    return list(permutations(elements))
```

In summary, Big O notation helps us analyze and compare the efficiency of algorithms by focusing on their growth rates relative to the input size. It enables us to identify algorithms that scale well for larger inputs and avoid those with poor time complexity for larger datasets.

OOP:

- What is a *mutator* method (also known as a "setter")?
  - The mutator method sets a value to a variable which allows it to be modified by external code.
  - Ex:

```
            # Mutator method to set the person's age
            def set_age(self, new_age):
                if new_age >= 0:
                    self.__age = new_age
                else:
                    print("Age must be a non-negative value.")
        ```
- What is an *accessor method* (also known as a "getter")?
    - The accessor method is a way of accessing the values of the object while ensuring the code cannot be changed.
        - Ex:

        ```
            # Accessor method to get the person's age
            def get_age(self):
                return self.__age
        ```
- What is a *constructor*?
    - An example of a constructor is "__init__", which is where you can set the initial values of the object's attributes.
    - Ex:

        ```
        class MyClass:
            def __init__(self, param1, param2, ...):
                # Initialize attributes here
                self.attribute1 = param1
                self.attribute2 = param2
                # ...

        # Creating an object of MyClass and invoking the constructor
        my_object = MyClass(value1, value2, ...)
        ```
- Ex:

    ```
    class Clock:
        def __init__(self, hours, minutes):
            if not 1 <= hours <= 23:
                if not 1 <= minutes <= 59:
                    raise ValueError
            self.__hours = hours
            self.__minutes = minutes

        def __str__(self):
            am_pm = ''
    ```

```
        if self.__hours < 12:
            am_pm = 'AM'
            new_hour = self.__hours
        elif self.__hours >= 12:
            am_pm = 'PM'
            new_hour = self.__hours - 12
        return f"The time is {new_hour}:{self.__minutes} {am_pm}"
def main():
    now = Clock(15, 10)
    print(now)

main()
```

- A class is used to create objects that all have the same behavior.
- Encapsulation is the act of providing a public interface to users while hiding the implementation details. An object's instance variables store the data required for executing its methods
- A mutator method does not change the object on which it operates; it just returns the value of one or more instance variables

- LIST COMPREHENSION - example: `squares = [x**2 for x in range(1, 6)]`
- Missed qs:
- How many elements (at most) does a binary search algorithm examine if a sorted list contains 88 values? Assume the value being searched for is NOT in the list. Explain your reasoning.
    - The list is sorted and the element being searched for is not in the list. That's log base 2 of 88 which is 6.45 which rounds to 7.
-