What will each line of the following code print?
Variables
What does the following code print?

```python
print(1+1)
print(str(1)+str(1))
print(str(1)+1)
print("Hello"*2)
```

Is this code equivalent to the code above?

```python
foo = 20
bar = foo*2
print(bar)
```

Is there anything wrong with this code?

```python
foo = 20
print(foo*2)
```

User Input
Is there anything wrong with the code below? Will it run? Will the answer be logically correct?

```python
# double the user's input
myInput = input("Please input a number here: ")
print(myInput*2)
```

Branching Control Structures
What is the output of the following code?
How could you make the code below more efficient?

```python
a = 25
b = False
if a < 25 or not b:
    b = not b
elif a >= 25 and b:
    a /= 5
else:
    a += 7
    b = False
print(f"{a}: {b}"
```

Loopying Control Structures

Fibonacci Sequence

We define the $i^{th}$ number of the Fibonacci sequence, $f_i$, to be $f_{i-1} + f_{i-2}$.

1. Write the code to find $f_i$ using a for loop.
2. Write the code to find $f_i$ using a while loop.
3. Which of these looping styles was more effective to solve the problem? Why? In what situations could the other style have been better?

Fizz Buzz

Write code that prints the first 100 numbers on individual lines. However, for multiples of 3, print 'fizz' instead; for multiples of 5, print 'buzz' instead; and for multiples of 3 and 5, print 'fizzbuzz' instead. For example, the first part of the output should be similar to:

```bash
1
2
'fizz'
3
'buzz'
'fizz'
7
```
Functions

Writing Functions

Are the following two functions equivalent?

```python
def foo(x):
    myVar = 2*x
    return myVar
def bar(x):
    return 2*x
```

What is the difference between the two functions below?

```python
def baz(x):
    return 2*x
def qux(x):
    print(2*x)
```

Using the same function definitions as above, identify the problem in the following code. Will it run? Will the result be what you expect? Why or why not?

```python
a = baz(3)
b = qux(3)
print(a)
print(b)
```

Identify the two problems in the following piece of code.

```python
def myFunc(x):
    x = int(input("Input any number."))
    if x > 10:
        return True
    return False
```

Square Root

Write a function that finds the square root of some integer $x$. This does not need to work for floats (though that would be great practice as well). You may return -1, "Does not exist", or raise an exception, if $x$ is not a perfect square.
Data Structures

Lists

Sum

Write a function, that takes a list as an argument, to find the sum of all elements in the list using a looping structure.

Exponent

Write a function, that takes a numeric value \( x \) (int or float), as well as an integer \( n \), and uses a looping structure to return the exponent \( x^n \).

Bonus: Can you change your function to work with any value of \( n \), int or float?

References

What outputs will the following blocks produce? What is the main difference?

```python
L = [1, 2, 3]

def doubleList(myList):
    for i in range(len(myList)):
        myList[i] *= 2
doubleList(L)
print(L)
```

```python
L = [1, 2, 3]

def doubleList(myList):
    for ele in myList:
        ele *= 2
doubleList(L)
print(L)
```

2-D Lists

Write a function which returns an \( n \times n \) 2-D list of randomly generated numbers between 0 and 100. Take \( n \) as an argument to your function.
Then, write two functions to print this 2-D list, line by line, as a string. The first function must be counter-controlled, whereas the second must use `foreach` loop(s). The functions must take two arguments: `grid`, your 2-D list, and `n`, the size of your `n` by `n` list. For example, with `n=2`:

```
27 4
46 9
```

**Bonus:** Starting with one of your two print functions, alter the code such that it only prints the values of the odd numbers, replacing all even numbers with `#`.

**Bonus:** Re-write your counter-controlled function to modify the 2D-list in-place such that it does not need to use `return` (as seen in the References section above).

### Dictionaries

When is a good time to use a dictionary? What are the advantages and disadvantages of using a dictionary?

Consider the following code.

```python
myDict = {}
for i in range(1, 5):
    myDict[i] = i*2

for key in myDict:
    print(f"{key}: {myDict[key]}")
```

Your friendly TA Jacob says:

I believe this code will print out "1:2, 2:4, 3:6, 4:8, 5:10", in that order, all on new lines.

However, your other friendly TA, Cameron, disagrees. Who is correct? If Cameron is correct, what are the reasons is Jacob wrong?

You are given the following code:

```python
prices = {
    "banana": 4,
    "apple": 2,
    "orange": 1.5,
    "pear": 3,
    "bread": 3,
    "bacon": 4,
    "chips": 2
}
```
Write a function, `receipt(userCart)`, to calculate the total cost of a user's cart and print a nicely formatted receipt. Assume that the `userCart` variable is a list of strings representing the items in the user's cart. For example, `["banana", "apple", "bacon", "apple"]` is a valid shopping cart.

Recursion

You may **not use any looping structures** for any of the questions in this section.

Tracing

We will calculate the sum of a list of numbers **recursively**. However, we will trace this by hand. You may also implement the code to do so if you wish. Complete the following trace:

```
sum [5,6,2,8,9]
5 + (sum [6,2,8,9])
5 + (6 + (sum [2,8,9]))
```

Now, we will do the same for the product of a list of numbers.

```
prod [10,9,6,4,2]
10 * # complete this line as well
```

Fibonacci

Recall the earlier question regarding the Fibonacci sequence. Instead of using a looping structure, we will rewrite this **recursively**.

1. What is the base case?
2. What is the recursive case?
3. Write the code to define the recursive function `fib(i)`.

Factorial

We define \( n! = n \cdot (n - 1) \cdot (n - 2) \cdots 2 \cdot 1 \).

1. What is the base case?
2. What is the recursive case?
3. Write the code to define the recursive function `fact(n)`.

   **Hint:** Your friendly TA Jacob suggests that because \( 4! = 4 \cdot (3 \cdot 2 \cdot 1) \), and \( (3 \cdot 2 \cdot 1) = 3! \), then we might say \( 4! = 4 \cdot 3! \).

Exponents

Write a recursive function `exp(x, n)` which returns the value \( x^n \).
Objects & Classes

Classes

What are classes? Name 3 advantages.

Implement an Animal class with the following specifications:

- Having the below attributes:
  - species
  - size
  - sound ("bark!")
  - isHungry (True or False)

- Having the below methods:
  - __init__(self, species, size, sound): A constructor to initialize these attributes for the animal. You may set isHungry to True or False as you please.
  - getSize(self): returns the size of the animal
  - makeNoise(self): prints the animal's sound
  - feed(self): if animal is hungry, change to false; if animal is not hungry, print something like: "Giraffe already ate!"

Then, implement a Zoo class with a list of Animal objects as an attribute, and an addAnimal(a) method which takes an animal as an argument and updates the list.

Bonus: Try changing the list of animal objects to a dictionary, using the objects as keys and the value as the count. Update your addAnimal(a) method accordingly.

Objects

What outputs will the following blocks of code produce?

```python
a = {1: 2, 2: 3}
b = {1: 2, 2: 3}

print(a == b)
print(a is b)
print(b is b)
```

```python
a = Animal("Dog", "Medium", "Bark!")
b = Animal("Dog", "Medium", "Bark!")
c = Animal("Dog", "Small", "Bark!")

print(a == b)
print(a is b)
print(b == c)
```
animalList = [a, b, c]
anotherAnimalList = [a, c, b]

print(animalList == anotherAnimalList)
print(animalList is anotherAnimalList)