The following exercises will give some practiceVariables.Run the following lines of python code in sage worksheet in cocalc. First predict the output, then write the actual output.

1. (a)
```
x = 5
print x
```

(b) Variables can be added, subracted, multiplied and divided. Also notice that they can be words, and need not just be single symbols.
```
x = 5
print x
y = 2
plus = x+y
times = x*y
divide = x/y
print plus
print times
print divide
```

(c) Having a variable to the left of an equals sign sets its value to the value of the right, even if that value is in terms of a variable.
```
x = 5
x = x + 2
print x
x = x + x
print x
```

(d) An important function in this class is the % symbol. Here $a\%b$ will return the remainder of a divided by b. Equivalently, it will return the element in $\mathbb{Z}/b\mathbb{Z}$ which is equivalent to a mod b.
```
print 5% 2
a = 2003
b = 1056
print a% b
print b% a
```

2. Lists are collections of variables. We can declare them (as empty lists) using the following line.

```
newList = []
```

(a) You can also initialize things nonempty.

```
newList = [2,4,6,7,10]
print newList
```

(b) To call an element of the list, simply use brackets to choose which number of the list you need (counting starts at 0!). Negative signs start counting from the back.

```
newList = [2,4,6,7,10]
print newList[0]
print newList[3]
print newList[-1]
print newList[2]+newList[4]
```

(c) You can change values of the list the same way you would change variables.

```
newList = [2,4,6,7,10]
print newList
newList[3] = 8
print newList
```

(d) The plus sign strings two lists together.

```
list1 = [2,4]
list2 = [6,7,10]
print list1 + list2
print list2 + list1
```

(e) You can use the above to add new elements to the list. You can also add things to the end of the list using append(). You can also compute the length of the list using `len`.

```
list1 = [2,4,6,7,10]
print len(list1)
list1 = list1 + [13]
print list1
print len(list1)
list1 = list1.append(16)
print list1
print len(list1)
```

(f) Write a program to do the following. First make a list of the first 3 primes, then a list of the first 3 even numbers. Print a list containing all six elements. Make a fourth list containing the sums of the elements in each list (i.e., the first element is the sum of the first even number and first prime, the second element is the sum of the second even and second prime, etcetera). Then print this list as well. Handwrite your code below (please check on cocalc that it does what we want it to do.)

3. If statements are extremely important logical components of writing code. The general idea is you write if statement: and then follow it with indented lines of code. The intended code will only run if the statement is true.

   (a) true will always be read as true. what about false?

   ```
   if true:
      print "True!"
   if false:
     print "False!"
   ```

   (b) We can put math statements in if statements. A single = sign is read as instantiation (setting a variable to something). The double equals sign == is read as a statement to check. We can also use

   ```
   if 2+2==4:
     print "True!"
   if 2+2==5:
     print "False!"
   ```

   (c) We can also use inequalities.

   ```
   if 2+2<4:
     print "2+2<4"
   if 2+2<=4:
     print "2+2<=4"
   if 2+2<5:
     print "2+2<5"
   ```

   (d) A common application in this class will be to check if things are equivalent mod certain moduli. The percent symbol will come in handy here! Notice that we will need it on both sides of the equation. Also notice variables and be used in if statements. Also notice that we can put variables and text in a print statement by separating them with commas.

   ```
   if 2 % 5 == 7 % 5:
     print "2 is equivalent to 7 mod 5"
   a = 6
   b = 22
   m = 8
   if a % m == b % m:
     print a, "is equivalent to", b, "mod", m
   ```

(e) A crucial application of if statements will be to search through lists. Indeed, we can just us an if statement to check if a certain number or string is in a list.

```
newList = [2,4,6,7,10]
if 7 in newList:
  print "7 is in the list"
if 8 in newList:
  print "8 is in the list"
else:
  print "8 is not in the list"
```

(f) Also notice the use of an else statement above. We can always follow an if statement with an else statement that will only run when the if statement is false.

```
x = 25
if x%2==0:
  print x, "is even"
else:
  print x, "is odd"
```

(g) As a final thing we can use between two statements in an if loop. It will be read as true if both statements are true.

```
x = 3
y = 5
if x<y && x+y % 2 == 0:
  print x,"is less than",y,"and",x,"plus",y,"is even".
```

4. Loops allow us to run computations over and over again.

   (a) While loops are similar to if statements, in that they take an input, and run the indented code if the input is true. The main difference is that and if statement will run the code once, a while loop will run over and over again until the statement isn't true. Explain what is happening.

```
x = 0
while x < 5:
  print "Still true"
  x = x+1
print "now it's false"
```

   (b) It is easy to get stuck in an infinite loop.

```
while true:
  print "Still true"
```

   (c) To escape from a loop, you could put a line that says break.

```
while true:
  print "Still true"
  break
```

   (d) We can use while loops in all sorts of ways. Below we combine a while loop print...what?

```
i = 0
while i<50:
  if i % 3 == 0:
    print i
  i = i+1
```

(e) Notice how it was useful to end a while loop by increasing i by one (thus preventing it from lasting forever). In fact, there is a type of loop that does that for us, called a for loop. The following code should do the same thing as above.

```
for i in range(0,50):
  if i % 3 == 0:
    print i
```

(f) A for loop runs through the i values in the range, including the lower bound, but excluding the upper bound. So below should print 0 through......what?

```
for i in range(0,10):
  print i
```

(g) A for loop can also run through a list.

```
myList = [2,4,6,7,10]
for i in myList:
  print i
```

(h) You can nest loops, meaning that you can run a loop inside of a loop.

```
for i in range(0,5):
  for j in range(0,5):
    print "i is ",i," and j is ",j
```

(i) You can use loops to populate lists. The following makes a list of the numbers less than 30 which are equivalent to 2 mod 5.

```
newList = []
for i in range(0,30):
  if i%5==2:
    newList.append(i)
print newList
```

5. Python (and sage) have many built in functions, but an amazing utility is that you can write your own. Do do so you define it using def, then name your function, and in parentheses, state your desired inputs.

```
def functionName(input1,input2):
```

This works in the following way. Whenever somewhere in your code a line says functionName(a,b), it will run the code indented below functionName, and will use a as input1 and b as input2. If there are no inputs, you just can leave empty parentheses:

```
def functionName():
```

(a) The following function prints Hello World. Notice that when we run the code nothing happens. Why? Add a line so that the printing actually happens, and make sure to check it runs.

```
def printHelloWorld():
  print "Hello World!"
```

(b) The function in part (a) just ran a line, but much like the functions we studied in calculus, a function can have a value, determined by the input. For instance, the following function computes $f(x, y) = x^2 + 2xy$.

```
def f(x,y):
  value = x*x + 2*x*y
  return value
z = f(2,4)
print z
```

(c) Functions can be handed many different data types. If within the function we use a certain data type, we must make sure to hand that function the correct data point. The following function takes a list of numbers as input, and returns a new list of all the even numbers in the original list.

```
def findEvenNumbersInList(listOfNumbers):
  evenNumbers = []
  for i in listOfNumbers:
    if i%2 == 0:
      evenNumbers.append(i)
  return evenNumbers

newList = [2,4,6,7,10]
print findEvenNumbersInList(newList)
```

(d) Try running the function above for different lists. What happens when you input an single integer to the function?

6. Ok, so there are the basics. Lets now try a few exercises related to our course.
   (a) Write a function `getDivisors(n)` whose input is an integer n, and whose output is list of all the (positive) divisors of n.

   (b) Write a function `isPrime(n)`, whose input is an integer n, and the output is `true` if n is prime and `false` otherwise. The `getDivisors(n)` function may come in handy here.

   (c) Write a function `getCommonDivisors(a,b)` which takes two integers a and b an returns a list of all of their common divisors.

   (d) Using the `getCommonDivisors(a,b)` function, write a `findGCD(a,b)` function which returns the greatest common divisor of two integers. Use this on homework problem 1.9. That is, the following lines of code should solve the problem for you:

```
print findGCD(291,252)
print findGCD(16261,85652)
print findGCD(139024789,93278890)
print findGCD(16534528044,8332745927)
```

7. This problem is somewhat more difficult then what we have seen so far, but the goal is to implement the euclidean algorithm and extended euclidean algorithm

   (a) Write a function `longDivision(a,b)` which takes as input two integers a and b, and returns a list `[quotient,remainder]` of the quotient and remainder after doing long division.

   (b) Use this function to implement the Euclidean algorithm to find the gcd of two numbers. Recall that you run longDivision over and over again, and the last nonzero remainder is the gcd. Run the same 4 lines from problem 6(d), and notice how much faster it goes! (Note, you really only need the remainder to run the Euclidean algorithm, so you could write this only using the % function rather than the long division function above. Either way is fine with me.)

   (c) Similarly, implement the extended Euclidean algorithm to find the u and v such that $a * u + b * v = gcd(a, b)$.

   (d) BONUS: Using the extended Euclidean algorithm, we can write a function `findInverse(a,m)`, which should take a modulus m, and an element $a \in \mathbb{Z}/m\mathbb{Z}$, and return the inverse of a in $\mathbb{Z}/m\mathbb{Z}$ if it exists.