Python Practice Worksheet

If you haven't done much programming before, the following exercises will give some practice of the basics, in particular with assigning, referencing, and manipulating variables. Run the given lines of code in a Jupyter notebook in cocalc. Try to predict the output before running the box. Also play around, see what happens when you do different things.

1. (a) You can assign numerical values to variables. The `print` command is useful for keeping track of what you are doing. `x = 5`

```
print(x)
```

(b) Variables can be added, subtracted, multiplied, and divided. Notice that they can be words, not just single symbols. `x = 5`

```
print(x)
y = 2
plus = x+y
times = x*y
divide = x/y
print(plus)
print(times)
print(divide)
```

(c) A variable can reference and even reassign itself.

```
x = 5
print(x)
x = x + 2
print(x)
x = x + x
print(x)
```

(d) An important function in this class is the % symbol, where $a\&b$ returns the remainder of a when divided by b.

```
print(5%2)
a = 2003
b = 1056
print(a%b)
print(b%a)
```

2. Variables can store more than numbers, for example, they can store lists of things, even lists of variables! (Lists are also often called arrays). You can initialize a (empty) list using the following line.
```
newList = []
```

   (a) You can also initialize nonempty lists.
   ```
   newList = [2,4,6,7,10]
   print(newList)
   ```

   (b) To reference an element of a list, simply use brackets to select the index of the list you'd like to reference. It starts counting at 0, and negative signs count from the back.
   ```
   newList = [2,4,6,7,10]
   print(newList)
   print(newList[0])
   print(newList[3])
   print(newList[-1])
   x = newList[2] + newList[4]
   print(x)
   ```

   (c) You can change values of a list the same whay 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 method above to add new elements to a list. You can also add things to the end of a list using `append()`. You can compute the length of alist using `len`.
   ```
   newList = [2,4,6,7,10]
   print(len(newList))
   newList = newList + [13]
   print(newList)
   print(len(newList))
   newList.append(16)
   print(newList)
   print(len(newList))
   ```

   (f) Write a program that does 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 6 elements. Make a fourth list consisting of the sums of the elements in each list (i.e., the first element is the first even number plus the first prime, etc.)and print it as well.

3. `if` statements are extremely important logical components of writing code. The general idea is that you write `if "...":`, where `...` is some logical statement, and then you follow the colon with some lines of code, which will run if and only if the `...` is true.

   (a) `True` will always be read as true. What about `False`?
   ```
   if True:
     print("True!")
   if False:
     print("False!")
   ```

   (b) We will often use mathematical statements in an `if` statement. We saw above that a single = sets variables, so we use == to check if a statement is true.
   ```
   if 2+2==4:
     print("2+2=4")
   if 2+2==5:
     print("2+2=5")
   ```

   (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. Finally, 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 is 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) Note 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) We can use two statements in an `if` statement, which, with either `and` or `or`. Here's an example with `and`. Play around with `or` too! `x = 3`

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

4. Loops allow us to run computataions 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 an `if` statement will run the code once, a `while` loop will run over and over again until the statement isn't true.

```
x = 0
while x < 5:
  print(x,"is less than 5")
x = x+1
print("now",x,"is not less than 5")
```

   (b) It is easy to get stuck in an infinite loop. `while True:`

```
  print("Still true")
```

   (c) The `break` command helps escape from loops.

```
while True:
  print("Still true")
  break
```

   (d) We can use `while` loops in all sorts of ways. Below we us a `while` loop and an `if` statement to print...what? `i = 0`

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

   (e) 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?