Jordan Boyd
PHY 494
HW2

2.1

a)

Git can help to keep track of changes throughout a project, making it easy to go backwards and forwards as necessary. If  the path the project takes ends up as a dead end the project can be easily reversed by looking at previous versions. It can also help when many people are working on a single project because it can keep track of everyone's changes and notify them when there is a conflict between them. These conflicts can then be easily fixed compared to having multiple different versions of one project across many different computers. Another thing Git helps with is that if an online repository is set up for the code then it makes it very easy for one user to work on the project from any computer. This makes it very easy to travel without carrying around a big computer everywhere.

b)

*git init* begins a new git repository in the current directory. *git clone* copies a repository from somewhere else into the current directory.

c)

*git commit* adds the last changes to the local repository while *git push* sends those changes to a remote repository.

2.2
My github username is *jbboyd*.

2.3

a) For loop

Syntax***:***

```
for <index> in range(<some range, e.g., 0 – 3>):
        <do this code>
```

Purpose: A for loop is used to loop through code a number of times set by the <index>

Example:

```
for i in range(5, 10):
   print(i)
```

b) While loop

Syntax:

```
while <statement = true>:
        <do this>
```

Purpose: A while loop is used to loop through code as long as the conditions used to set up the loop are true.

Example:

```
i = 0
while (i < 10):
        i += 1
        print(i)
```

c) if statement

Syntax:

```
if (<boolean expression>):
   <do this>
elif (<another boolean expression>):
   <do this instead>
else:
   <do this other thing>
```

Purpose: An if statement is used to check for certain conditions. If those conditions are met then the code inside the statement is run. An elif can also be added as a second check if the first conditions aren't met. If the conditions of the elif are met then the code inside that statement runs. Last, an optional else statement can be added that will cause the code inside to run if the original if statement or optional elif statement conditions aren't met.

*Example:*

```
A = 1
B = 2
C = 2
if (A == B):
print(A + B)
elif (B == C):
print(B + C)
```

d) break statement

Syntax:

```
<some loop>:
        <some code>
```

```
          if (<boolean expression>):
                    break
```

Purpose: A break statement is used to stop a loop and continue from the next statement after the loop. It is useful as a way to stop the loop without meeting the conditions that would normally stop the loop. It can be a useful way to make sure a loop doesn't run forever.

Example:

```
loopCatcher = 0
while (2 ==2):
          loopCatcher += 1
          if (loopCatcher > 1000):
                    break
print(loopCatcher)
```

e) continue statement

Syntax:

```
<some loop>
          if (<boolean expression>):
                    <do this>
                    continue
          <do something else>
```

Purpose: The continue statement is used to send a loop to the next iteration without executing any of the code before the continue statement.

Example:

```
i = 1
A = 0
while (i < 30):
          if (i % 5 == 0):
                    A += i
                     i += 1
                     print(A)
                     continue
           i += 1
```

2.4

a) *bag[1:3]*

b) *bag[::-1]* prints the entire list in reverse order

c) *ga[0:4]*
*ga[15:20]*

d)

i) *bag[0] = 'book'* replaces the first item of *bag* with *book*.
ii) *bag = ['book', 'towel', 'tea', 'mice']*
*mybag = ['book', 'towel', 'tea', 'mice']*
*yourbag = ['book', 'towel', 'tea', '42', 'money']*
iii) *x = a* points each list to the same data, if you change item 3 for x, then you change it for a too. *y = a[:]* creates a new list for y so that any changes to *a* don't affect *y.*

e)

i) It causes an error that reads "*'str' object does not support item assignment*"
ii) *newga = "Three" + ga[4:]*

f) *ga.split()* breaks each word in *ga* into an individual string and stores it as an item in a list containing each of the strings.

*a, b, c = ga.split()[:3]* stores the first for words individually as strings in *a, b,* and *c,* respectively.

*list([1, 2, 3])* creates a list containing 1, 2, and 3 as separate items.

*list([ga])* creates a list containing each character, including spaces of *ga* as a separate item.

g)

i) *bags[0][:]*
ii) *bags[0][1]*
iii) *bags[1][2]*

2.5

a) *positions[1]* is the location of the second particle, the output is [1.34234, 1.34234, 0.0]

b) *positions[1][1]* gives the y-coordinate of the second particle. The output is 1.34234.

c)

```
for i in range(0, 4):
        for j in range(0, 3):
                positions[i][j] += t[j]
```

The output is below:
*[[1.34234, -1.34234, -1.34234], [2.68468, 0.0, -1.34234], [2.68468, -1.34234, 0. 0], [1.34234, 0.0, 0.0]]*

d) The code is:

```
def translate(coordinates, t):
        """Translates a given list of coordinates by vector t"""
        for i in range(0,len(coordinates)):
                for j in range(0, 3):
                        particles[i][j] += t[j]
        return coordinates

translate(positions, t)
```

The output for positions and t is:

```
[[1.34234, -1.34234, -1.34234],  [2.68468, 0.0, -1.34234],  [2.68468, -1.34234, 0.
0],  [1.34234, 0.0, 0.0]]
```

For positions2 it's:

```
translate(positions2, t)

[[2.84234, -2.84234, 1.65766], [-0.15765999999999999, -2.84234, -4.34234]]
```