Pair Programming Exercise, flow control, if/esle, Recursion

For DSE5002, Module 3, PETER GYORDA APRIL 4, 2025

If, else statements

Recursion

Loops

See discussion in Think Python

https://allendowney.github.io/ThinkPython/chap05.html - ifs and test conditions, recursion

https://allendowney.github.io/ThinkPython/chap07.html, Iteration, ie looops

If

Python has an if statement, which executes a code block if it is true -code blocks are denoted by indentations, each level of a code block should be indented by 4 spaces 4 spaces is not strictly required, any consistent number will work, but for consistency use the default 4 spaces this will help a lot if someone else has to edit your code

Elif

to go with the if statement, there is an elif ("else if") command which uses a second test condition. If will run the associated code when the if was false and the elif is true

Else

this is the same as the else in R, it will run if the if and elif statements before it do not run

Simple if

```
In [1]:  x=5
         if x<=5:
             age="juvenile"
             print(age)
```

juvenile

here is an if-else pairing

```
In [2]:  x=6

         if x<=5:
             age='juvenile'
         else:
             age="adult"

         print(age)
```

adult

Question/Action

Alter the code above to produce the juvenile answer

```
In [3]:  x=1

         if x<=5:
             age='juvenile'
         else:
             age="adult"

         print(age)
```

juvenile

Question/Action

write code that will test to see if the variable home_state is "Ma" or not, and set y = 0 for "Ma" and y=1 otherwise

print out the value of y at the end

```
In [4]:  home_state = "Ma"   # You can change this value to test different scenarios
         y = 0

         if home_state == "Ma":
             y = 0
         else:
             y = 1

         print(y)
```

0

Using a chain of if, elif and else

Lets set y to be

1-for "Ma" 2-for "Ny" 3-for "Ct" 4 for all other states

```
In [5]:  home_state="Ne"

         if(home_state=="Ma"):
             y=1
         elif(home_state=="Ny"):
             y=2
         elif(home_state=="Ct"):
             y=3
         else:
             y=4

         print(y)
```

4

For Loop

when we use a for loop, we can loop through any iterable variable type, so a list,or tuple, but not a set

```
In [5]:  x=[1,2,3,4,5,6,7,8,9,10]

         for i in x:
             print(i, i**2)
```

```
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
10 100
```

```
In [6]:  #Iterating on a numpy matrix

         import numpy as np

         z=np.arange(15).reshape(5,3)

         z
```

```
Out[6]:  array([[ 0,  1,  2],
                [ 3,  4,  5],
                [ 6,  7,  8],
                [ 9, 10, 11],
                [12, 13, 14]])
```

```
In [7]:  #iterate on rows of an numply matrix

         nrows=z.shape[0]                              # shape gives use the dimension of the
```

```python
for k in np.arange(nrows):
    print(z[k,:])
```

```
[0 1 2]
[3 4 5]
[6 7 8]
[ 9 10 11]
[12 13 14]
```

In [20]:
```python
#iterate on columns of an numply matrix

ncols=z.shape[1]                              # shape gives use the dimension of the

for k in np.arange(ncols):
    print(z[:,k])
```

```
[ 0  3  6  9 12]
[ 1  4  7 10 13]
[ 2  5  8 11 14]
```

In [13]:
```python
y={1:"Bob",2:"Shauna",3:"Seung",4:"Jose"}

#we cannot loop directly on a dictionary,  but we can loop on the values

for value in y.values():
    print(value)

# we can iterate on both the key and the value, using the items function

for key,values in y.items():
    print(key,value)
```

```
Bob
Shauna
Seung
Jose
1 Jose
2 Jose
3 Jose
4 Jose
```

While Loop

Python also has a while loop, that continues until a max value is reached

In [30]:
```python
x=[0]
n=1

while(n<13):
    x.append(n**2)
    n=n+2

print(x)
```

```
[0, 1, 9, 25, 49, 81, 121]
```

In [ ]:
```
# Nested Loops


Given a list of values, which pairs of values in the list, if any, add up to a targ

Use two nested loops to compare the sums of all possible combinations in the list t
```

In [8]:
```python
x=[1,2,4,6,9,10,15,17,20,23,24]
target_sum=27

nsteps_nested=0

for i in np.arange(len(x)):
    for j in np.arange(len(x)-1):
        nsteps_nested=nsteps_nested+1
        if ((x[i] != x[j+1]) & ((x[i] + x[j+1]) == target_sum)):
            print(x[i], x[j+1])

print()
print(nsteps_nested)
```

```
4 23
10 17
17 10
23 4

110
```

In [ ]:
```
# Hashed based method,   allows us to use only one loop iteration

The python dictionary is hashed storage

So what will do is set up an entry dictionary,

then we will loop through x once,

For a given value in x,  x[i], we will use vector operations to find if there is a
that sums with x[i] to the target_sum,   if there is, we will use x[i] as the key,
as the value in the dictionary.

After one loop iteration, the key and value pairs of the dictionary hold the paired
```

In [66]:
```python
x=np.array([1,2,4,6,9,10,15,17,20,23,24])
target_sum=27

nsteps_hash=0

h={}    # an empty dictionary

for i in np.arange(len(x)):
    nsteps_hash=nsteps_hash+1
    if any((x[i]+x)==target_sum)&(x[i]!=target_sum/2):
        h[x[i]]=(x[(x[i]+x)==target_sum])[0]
```

```
print(h.items())
print()
print(nsteps_hash)
```

dict_items([(4, 23), (10, 17), (17, 10), (23, 4)])

11

Notice that the number of loop iterations for the nested loop was 110, and the number of loop operations in the hashed or dictionary version was 11.

In a large problem, this difference would really add up.

To figure out how to cover a nested loop to a single loop with a hash (dictionary)

a.) Loop at the inner loop and figure out how to vectorize that operation

b.) Set up an empty dictionary

c.) set up the outer loop, and put the vectorized operation within the outer loop. For each step of the outer loop, store the key value pairing from your vectorized calculation into the dictionary. Use the value from the outer loop as the key, and other value, or the calculated value as the value in the dictionary.

In [9]:
```python
import numpy as np

x = np.array([1, 2, 4, 6, 9, 10, 15, 17, 20, 23, 24])
target_sum = 27

nsteps_hash = 0
h = {}

for i in np.arange(len(x)):
    nsteps_hash = nsteps_hash + 1
    complement = target_sum - x[i]

    if complement in h:
        if x[i] < complement:
            h[x[i]] = complement
        elif complement < x[i]:
            h[complement] = x[i]
    elif x[i] != complement:
        h[x[i]] = None

print(h.items())
print()
print(nsteps_hash)
```

dict_items([(1, None), (2, None), (4, 23), (6, None), (9, None), (10, 17), (15, None), (20, None), (24, None)])

11

Recursion

This is a method in which a function calls itself repeatedly

One classical example is to compute n! (n factoriaL) by repeatedly calling a function.

The function returns 1 when n=0, 0! is defined as 1.

For other values, it computes N x (N-1)! by recursively calling the factorial fucntion

Note: There is a lot of overhead involved in calling recursive functions

```
In [10]: def rec_factorial(n):
             if n==0:
                 return(1)
             else:
                 return(n*rec_factorial(n-1))
```

```
In [11]: rec_factorial(2)
```

Out[11]: 2

```
In [ ]: ## *Question/Action*

        Write a function that recursively counts down toward zero

        If n is zero, it should print zero

        Otherwise it should print n with a linefeed and then call itself again with an inpu
```

```
In [12]: def countdown_recursive(n):
           if n == 0:
             print(0)
           else:
             print(n)
             countdown_recursive(n - 1)

         countdown_recursive(5)
```

```
5
4
3
2
1
0
```

In [ ]: