Unit 14

Loops & List Comprehensions

Asg 14.4 (Coding)

while Loops; Validating Input; Sentinel Values;

Break, Continue, and Pass Statements; and Indefinite Iteration

In [1]:
```
# set up notebook to display multiple output in one cell

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

print('The notebook is set up to display multiple output in one cell.')
```

```
The notebook is set up to display multiple output in one cell.
```

Problem #1:

Write code that uses a while loop to print out the first 20 odd integers in a single row with a space between each integer.

Desired Output

```
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41
```

In [2]:
```
x = 0
while x < 42:
    if x %2==0:
        x+=1
        continue
    else:
        print(x, end=" ")
        x+=1
```

```
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41
```

Problem #2:

Start with the string 'Brookfield' and use a while loop to strip away characters from the string one at a time, starting with the first character until the string is empty (which will then end the loop). At each stage, print out the part of the string that remains, making sure to put two spaces between each part of the string that you print out.

Desired Output

```
Brookfield  rookfield  ookfield  okfield  kfield  field  ield  eld  ld  d
```

In [3]:
```
string = "Brookfield"
counter = 0
while counter < len(string):
    print(string[counter:len(string)],end=" ")
    counter+=1
```

```
Brookfield rookfield ookfield okfield kfield field ield eld ld d
```

Problem #3:

Write code that uses a nested while loop to display all possible ordered pairs (tuples) with x-values of 1 through 6 paired with y-values of 1, 2, and 3. Make sure that there is a space between ordered pairs that you print out.

Desired Outsput

```
(1, 1) (1, 2) (1, 3) (2, 1) (2, 2) (2, 3) (3, 1) (3, 2) (3, 3) (4, 1) (4, 2) (4, 3) (5, 1) (5, 2) (5, 3) (6, 1) (6, 2) (6, 3)
```

In [6]:
```
x = 1
y =1
while x in range(1,7):
    while y in range(1,4):
        print(f"({x},{y})",end=" ")
        y+=1
    y=1
    x+=1
```

```
(1,1) (1,2) (1,3) (2,1) (2,2) (2,3) (3,1) (3,2) (3,3) (4,1) (4,2) (4,3) (5,
1) (5,2) (5,3) (6,1) (6,2) (6,3)
```

Problem #4:

Write code that uses a nested while loop to display the desired output:

Desired Outsput

```
1 1 1 1 1 1
2 2 2 2 2
3 3 3 3
4 4 4
5 5
6
```

In [13]:
```
x = 6
y = 1
while x > 0:
    while y < 8-x:
        print(str(y) * x)
        y += 1
    x-=1
```

```
111111
22222
3333
444
55
6
```

Problem #5:

Write code that uses a while loop to sum the elements in the list [11, 32, 53, 85, 127,159]. Include a formatted print statement to print out the sum.

Desired Outsput

```
The sum of the numbers in the list [11, 32, 53, 85, 127, 159] is 467.
```

In [19]:
```
list = [11,32,53,85,127,159]
total  = 0
x = 0
while x < len(list):
    total += list[x]
    x+=1
print(f"The sum of the numbers in the list {list} is {total}.")
```

```
The sum of the numbers in the list [11, 32, 53, 85, 127, 159] is 467.
```

Problem #6:

Write code that uses a while loop to print out the numbers between 1 and 250 that have whole number square roots. Make sure to print out the output in a single row with a space between each whole number square root.

Desired Outsput

```
1 4 9 16 25 36 49 64 81 100 121 144 169 196 225
```

In [22]:
```
x = 1
while (x**2) < 250:
    print(x**2, end=" ")
    x+=1
```

```
1 4 9 16 25 36 49 64 81 100 121 144 169 196 225
```

Problem #7:

Create a program that incorporates a while loop and prompts a user for their name and for their test scores. Continue to prompt the user for additional test scores until the user types 'q'. When the user types 'q', the program stops asking the user for test scores and outputs a formatted print statement identical to what is indicated in the desired output found below.

Use the following inputs for your test case:

Name = sally

1st test score = 87 2nd test score = 96 3rd test score = 91 4th test score = 89

Desired Outsput

```
What is your name?  sally
Enter your first test score:  87
Enter your next test score or q if there are no more test scores to enter:  96
Enter your next test score or q if there are no more test scores to enter:  91
Enter your next test score or q if there are no more test scores to enter:  89
Enter your next test score or q if there are no more test scores to enter:  q

Sally, your test score average is 90.75%.
```

In [25]:
```
name = input("what is your name? ")
fts = int(input("Enter your first test score: "))
counter = 1
nts = int(input("Enter your next test score or q if there ar eno mroe test sc
while nts != "q":
    fts += nts
    counter += 1
    try:
        nts = int(input("Enter your next test score or q if there ar eno mroe
    except:
        break
print(f"{name.capitalize()}, your test score average is {fts/counter:.2f}%")
```

```
what is your name? Roheat
Enter your first test score: 96
Enter your next test score or q if there ar eno mroe test scores to enter: 9
1
Enter your next test score or q if there ar eno mroe test scores to enter: 8
9
Enter your next test score or q if there ar eno mroe test scores to enter: q
Roheat, your test score average is 92.00%
```

Problem #8:

Write code that uses a while loop , the multiplication operator, *, and the tuple our_tuple = ('Pandas', 'Numpy', 'Seaborn', '!', 7) to produce the output found below:

Desired Output

```
PandasPandasPandas
NumpyNumpyNumpy
SeabornSeabornSeaborn
!!!
21
```

In [26]:
```
our_tuple = ('Pandas', 'Numpy', 'Seaborn', '!', 7)
counter = 0
while counter < len(our_tuple):
    print(our_tuple[counter]*3)
    counter += 1
```

```
PandasPandasPandas
NumpyNumpyNumpy
SeabornSeabornSeaborn
!!!
21
```

Validating Input

You can also use a while loop when you want to validate input; when you want to make sure the user has entered valid input for a prompt.

Problem #9:

Use a while loop to validate user input. Ask the user to enter a day of the week. Keep asking the user for a day of the week until they enter one. When the user enters a day of the week, output a formatted print statement identical to that in the desired output found below.

For a test case, use the following answers:

1st Answer: October 2nd Answer: November 3rd Answer: Tuesday

Desired Outsput

```
Please enter a day of the week:  October
Please enter a day of the week:  November
Please enter a day of the week:  Tuesday

Yes! You did it!! Tuesday is a day of the week.
```

In [28]:
```
days = ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', '
a = input("please enter a day of the week: ")
while a.lower() not in days:
    a = input("please enter a day of the week: ")
print(f"Yes! You did it!! {a} is a day of the week.")
```

```
['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunda
y']
please enter a day of the week: October
please enter a day of the week: November
please enter a day of the week: Tuesday
Yes! You did it!! Tuesday is a day of the week.
```

Sentinal Values

Indefinite loops are much more common in the real world than definite loops.

- If you are selling tickets to an event, you don't know in advance how many tickets you will sell. You keep selling tickets as long as people come to the door and there's room in the hall.
- When the baggage crew unloads a plane, they don't know in advance how many suitcases there are. They just keep unloading while there are bags left in the cargo hold. (Why your suitcase is always the last one is an entirely different problem.)
- When you go through the checkout line at the grocery, the clerks don't know in advance how many items there are. They just keep ringing up items as long as there are more on the conveyor belt.

Let's implement the last of these in Python, by asking the user for prices and keeping a running total and count of items. When the last item is entered, the program gives the grand total, number of items, and average price. We'll need these variables:

- **total** - this will start at zero
- **count** - the number of items, which also starts at zero
- **moreItems** - a boolean that tells us whether more items are waiting; this starts as True

The pseudocode (code written half in English, half in Python) for the body of the loop looks something like this:

This pseudocode has no option to set **moreItems** to False, so it would run forever. In a grocery store, there's a little plastic bar that you put after your last item to separate your groceries from those of the person behind you; that's how the clerk knows you have no more items. We don't have a "little plastic bar" data type in Python, so we'll do the next best thing: we will use a price of zero to mean "this is my last item." In this program, zero is a **sentinel value**, a value used to signal the end of the loop.

Problem #10:

Write code to produce the desired output found below for a shopping order that consists of the following purchases:

Purchases

1st Purchase = $51.78
2nd Purchase = $79.11
3rd Purchase = $124.56

Desired Outsput

```
Enter price of item (0 when done): $51.78
Subtotal: $51.78
Nummber of items: 1

Enter price of item (0 when done): $79.11
Subtotal: $130.89
Nummber of items: 2

Enter price of item (0 when done): $124.56
Subtotal: $255.45
Nummber of items: 3

Enter price of item (0 when done): $0

Total number of items: 3
Total Price: $255.45
Average price per item: $85.15
```

In [29]:
```
subtotal = 0.0
i = 0
price = float(input("Enter price of item (0 when done): "))
while price != 0:
    subtotal += price
    i += 1
    print(f"Subtotal: ${subtotal:.2f}")
    print(f"Number of items: {i} \n")
    price = float(input("Enter price of item (0 when done): "))


print(f"Total number of items: {i}")
print(f"Total Price: ${subtotal:.2f}")
print(f"Average price per item: ${subtotal/i:.2f}")
```

```
Enter price of item (0 when done): 51.78
Subtotal: $51.78
Number of items: 1

Enter price of item (0 when done): 79.11
Subtotal: $130.89
Number of items: 2

Enter price of item (0 when done): 124.56
Subtotal: $255.45
Number of items: 3

Enter price of item (0 when done): 0
Total number of items: 3
Total Price: $255.45
Average price per item: $85.15
```

How To Use Break, Continue, and Pass Statements when Working with Loops

Read the following article before doing the remaining questions.

Python Break, Continue, and Pass (https://pynative.com/python-break-continue-pass/)

Problem #11:

Write code that uses a while loop that is designed to print out the first 30 integers, but that also incorporates a break statement to terminate the program if it reaches a number that has both 3 and 4 as factors. Make sure that your output prints out as a single row with a space between each number in the row as illustrated in the desired code.

Desired Output

```
1 2 3 4 5 6 7 8 9 10 11
```

In [2]:
```
i = 1
while i <= 30:
    if i % 3 == 0 and i % 4 == 0:
        break
    print(i, end = " ")
    i += 1
```

```
1 2 3 4 5 6 7 8 9 10 11
```

Problem #12:

Write code that uses a while loop that is designed to print out the first 30 integers, but that incorporates a continue statement to omit any number that has both 2 and 3 as factors. Make sure that your output prints out as a single row with a space between each number in the row as illustrated in the desired code.

Desired Output

```
1 2 3 4 5 7 8 9 10 11 13 14 15 16 17 19 20 21 22 23 25 26 27 28 29
```

In [3]:
```
counter = 0
while counter < 30:
    counter += 1
    if counter % 2 == 0 and counter % 3 == 0:
        continue
    print(counter, end = " ")
```

```
1 2 3 4 5 7 8 9 10 11 13 14 15 16 17 19 20 21 22 23 25 26 27 28 29
```

Problem #13:

Write code that uses a while loop and a for loop and that is designed to print out all of the prime numbers less than or equal to 100, but that incorporates a break statement to terminate the program after the first non-prime number is reached. Make sure that your output prints out as a single row with a space between each number in the row as illustrated in the desired code.

Desired Output

```
1 2 3
```

In [4]:
```
i = 1
while i <= 100:
    flag = True
    for j in range(2,i):
        if i%j == 0:
            flag = False
    if flag:
        print(i, end = " ")
    else:
        break
    i+= 1
```

```
1 2 3
```

Problem #14:

Write code that uses a while loop and a for loop and that is designed to print out all of the prime numbers less than or equal to 100, and that incorporates a continue statement to omit any non-prime numbers. Make sure that your output prints out as a single row with a space between each number in the row as illustrated in the desired code.

Desired Output

```
1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

In [6]:
```
i = 0
while i < 100:
    i+= 1
    flag = True
    for j in range(2,i):
        if i%j == 0:
            flag = False
    if flag:
        print(i, end = " ")
    else:
        continue
```

```
1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Bonus Question:

Write code that uses the following rule for creating a sequence:

- Step 1: Start with some positive integer.
- Step 2: Generate the next term of the sequence from the positive integer from Step 1, either by halving that positive integer whenever it is even, or else by multiplying it by three and adding 1 when it is odd.
- Step 3: Have the sequence terminate when it reaches 1.

Run your code for the positive integers 5 through 10 and have your output print out the terms of the sequence along with the number of iterations required to reach termination for each of those positive integers as is shown in the desired output found below.

Desired Output

```
The integer that is being used to start the 3n + 1 sequence is n = 5.
The terms of the sequence are: 5 16 8 4 2 1
The number of iterations required to reach termination = 5.

The integer that is being used to start the 3n + 1 sequence is n = 6.
The terms of the sequence are: 6 3 10 5 16 8 4 2 1
The number of iterations required to reach termination = 8.

The integer that is being used to start the 3n + 1 sequence is n = 7.
The terms of the sequence are: 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
The number of iterations required to reach termination = 16.

The integer that is being used to start the 3n + 1 sequence is n = 8.
The terms of the sequence are: 8 4 2 1
The number of iterations required to reach termination = 3.

The integer that is being used to start the 3n + 1 sequence is n = 9.
The terms of the sequence are: 9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
The number of iterations required to reach termination = 19.

The integer that is being used to start the 3n + 1 sequence is n = 10.
The terms of the sequence are: 10 5 16 8 4 2 1
The number of iterations required to reach termination = 6.

This problem is a good example of INDEFINITE ITERATION.
```

In [*]:
```
integer = int(input("The integer that is being used to start the 3n + 1 seque
a = []

while integer:
    a.clear()
    a.append(integer)
    counter = 0
    while integer != 1:
        while integer % 2 == 0:
            integer /= 2
            integer = int(integer)
            a.append(integer)
            counter += 1
        while integer % 2 == 1 and integer != 1:
            integer = (integer * 3) + 1
            a.append(integer)
            counter += 1

    print("The terms of the sequence are: ", end = "")
    for number in a:
        print(number, end = " ")
    print(f" \nThe number of iterations required to reach termination = {coun
    integer = int(input("The integer that is being used to start the 3n + 1 s
```

```
The integer that is being used to start the 3n + 1 sequence is n = 5
The terms of the sequence are: 5 16 8 4 2 1
The number of iterations required to reach termination = 5

The integer that is being used to start the 3n + 1 sequence is n = 6
The terms of the sequence are: 6 3 10 5 16 8 4 2 1
The number of iterations required to reach termination = 8

The integer that is being used to start the 3n + 1 sequence is n = 7
The terms of the sequence are: 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
The number of iterations required to reach termination = 16

The integer that is being used to start the 3n + 1 sequence is n = 8
The terms of the sequence are: 8 4 2 1
The number of iterations required to reach termination = 3

The integer that is being used to start the 3n + 1 sequence is n = 9
The terms of the sequence are: 9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16
8 4 2 1
The number of iterations required to reach termination = 19

The integer that is being used to start the 3n + 1 sequence is n = 10
The terms of the sequence are: 10 5 16 8 4 2 1
The number of iterations required to reach termination = 6
```

```
The integer that is being used to start the 3n + 1 sequence is n =
```

In [ ]: