In [3]:
```
# Repeated Steps
n = 5
while n > 0 :
    print(n)
    n = n - 1
print('Blastoff!')
print(n)

# Loops (repeated steps) have iteration variables that change each time through a l
```

```
5
4
3
2
1
Blastoff!
0
```

In [3]:
```
# An Infinite Loop
n = 5
while n > 0 :
    print('Lather')
    print('Rinse')
print('Dry off!')
# What is wrong with this loop?
# it will never end because n will never become less than 0, thus it will go on inf
```

```
Lather
Rinse
Lather
Rinse
Lather
Rinse
Lather
Rinse
Lather
Rinse
Lather
Rinse
Lather
Rinse
Lather
Rinse
Lather
Rinse
Lather
Rinse
Dry off!
```

In [7]:
```
n = 0
while n > 0 :
    print('Lather')
    print('Rinse')
print('Dry off!')
# What is this loop doing?
# The loop will not run because the condition will never be met.
```

```
Dry off!
```

In [11]:
```
# Breaking Out of a Loop
# • The break statement ends the current loop and jumps to the statement immediatel
# • It is like a loop test that can happen anywhere in the body of the loop
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
Hello There!
Finished!
Done!
```

In [13]:
```
# The CONTINUE statement ends the current iteration and jumps to the top of the loo
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')

# Indefinite Loops
# • While loops are called “indefinite loops” because they keep going until a logic
# • The loops we have seen so far are pretty easy to examine to see if they will te
# • Sometimes it is a little harder to be sure if a loop will terminate
```

```
Hello There!
Print this
Done!
```

In [17]:
```
# Definite Loops
# Iterating over a set of items…
# • Quite often we have a list of items of the lines in a file -effectively a finit
# • We can write a loop to run the loop once for each of the items in a set using t
# • These loops are called “definite loops” because they execute an exact number of
# • We say that “definite loops iterate through the members of a set”
```

In [19]:
```
# A Simple Definite Loop
for i in [5, 4, 3, 2, 1] :
    print(i)
print('Blastoff!')
# The iteration variable moves through all of the values inthe sequence
# Definite loops (for loops) have explicit iteration variablesthat change each time
```

```
5
4
3
2
1
Blastoff!
```

In [21]:
```
# A Definite Loop with Strings
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends :
    print('Happy New Year:', friend)
print('Done!')
```

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!
```

In [23]:
```
print('Before')
for thing in [9, 41, 12, 3, 74, 15] :
    print(thing)
print('After')
```

```
Before
9
41
12
3
74
15
After
```

In [25]:
```
# Finding the Largest Value
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)
print('After', largest_so_far)
```

```
Before -1
9 9
41 41
41 12
41 3
74 74
74 15
After 74
```

In [27]:
```
# Counting in a Loop
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + 1
    print(zork, thing)
print('After', zork)
# To count how many times we execute a loop, we introduce a counter variable that s
```

```
Before 0
1 9
2 41
3 12
4 3
5 74
6 15
After 6
```

In [29]:
```
# Summing in a Loop
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + thing
    print(zork, thing)
print('After', zork)
# To add up a value we encounter in a loop, we introduce a sum variable that starts
```

```
Before 0
9 9
50 41
62 12
65 3
139 74
154 15
After 154
```

In [35]:
```
# Finding the Average in a Loop
count = 0
sum = 0
print('Before', count, sum)
for value in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    sum = sum + value
    print(count, sum, value)
print('After', count, sum, sum / count)
# An average just combines the counting and sum patterns and divides when the loop
```

```
Before 0 0
1 9 9
2 50 41
3 62 12
4 65 3
5 139 74
6 154 15
After 6 154 25.666666666666668
```

In [37]:
```
# Filtering in a Loop
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if value > 20:
        print('Large number',value)
print('After')
# We use an if statement in the loop to catch / filter the values we are looking fo
```

```
Before
Large number 41
Large number 74
After
```

In [39]:
```
# Search Using a Boolean Variable
found = False
print('Before', found)
for value in [9, 41, 12, 3, 74, 15] :
    if value == 3 :
        found = True
    print(found, value)
print('After', found)
# If we just want to search and know if a value was found, we use a variable that s
```

```
Before False
False 9
False 41
False 12
True 3
True 74
True 15
After True
```

In [43]:
```
# How to Find the Smallest Value?
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)
print('After', largest_so_far)

# How would we change this to make it find the smallest value in the list?
smallest_so_far = -1
print('Before', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)
print('After', smallest_so_far)
# We switched the variable name to smallest_so_far and switched the > to <
```

```
Before -1
9 9
41 41
41 12
41 3
74 74
74 15
After 74
Before -1
-1 9
-1 41
-1 12
-1 3
-1 74
-1 15
After -1
```

In [45]:
```python
# Finding the Smallest Value
smallest = None
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)
print('After', smallest)
# We still have a variable that is the smallest so far. The first time through the
```

```
Before
9 9
9 41
9 12
3 3
3 74
3 15
After 3
```

In [47]:
```python
# The is and is not Operators
smallest = None
print('Before')
for value in [3, 41, 12, 9, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)
print('After', smallest)
# • Python has an is operator that can be used in logical expressions
# • Implies “is the same as”
# • Similar to, but stronger than ==
# • is not also is a logical operator
```

```
Before
3 3
3 41
3 12
3 9
3 74
3 15
After 3
```

In [ ]: