Python Loops

Two types of `for` loops

- iterating through a numeric range
- iterating through a container
- C-style for loop (`for (c = 0; c < 10; c++)`) is not supported

In [2]:
```
for num in range(25):
    print(num, end=' ')
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

In [2]:
```
mylist = 'small medium large'.split()
for size in mylist:
    print(size)
```

```
small
medium
large
```

Looping over strings

- Able to loop over strings as well
- Allows processing character-by-character

In [3]:
```
name = 'Colonel Sanders'
for c in name:
    print(c, end = '-')
```

```
C-o-l-o-n-e-l- -S-a-n-d-e-r-s-
```

`break` and `continue`

- In some cases, you may want to "short-circuit" a loop (or an iteration)
- `break` can be used to exit from a loop, moving to statements beyond
- `continue` can be used to exit from an iteration of a loop, moving to next iteration

In [3]:
```
movies = ['Star Wars', 'The Goonies', 'The Godfather', 'Wizard of Oz', 'Dumb & Dumber', 'Forest Gump']
for movie in movies:
    if movie == 'The Godfather':
        print(movie, 'the GOAT')
        continue
    elif movie == 'Dumb & Dumber':
        break
    else:
        print(movie)
```

```
Star Wars
The Goonies
The Godfather the GOAT
Wizard of Oz
```

`range` function

- To iterate over numbers in a sequence, can use a list specifying each number
- Alternatively, you can use `range()` as a shortcut
- Format is `range([s, ]e[, step])`
- Starting value and step are optional
- Starting value defaults to 0 if not included
- If step is omitted, increments by 1; otherwise, increments by provided step
- Resulting list will include values from starting value up to, but not including, ending value

In [4]:
```
for num in [0, 1, 2, 3, 4]:
    print(num, end = '')
print()
for num in range(5):
    print (num, end = '')
print()
for num in range(2, 10):
    print(num, end = '')
print()
for num in range(0, 10, 3):
    print(num, end = '')
print()
```

```
01234
01234
23456789
0369
```

`else`

- Can use an `else` statement to define code to be executed on completion of the loop
- If you break out of the loop, the `else` won't get executed

In [6]:

```
for num in [0, 1, 2, 3, 4]:
    print(num, end = '')
else:
    print()
for num in range(5):
    print (num, end = '')
else:
    print()
for num in range(2, 10):
    print(num, end = '')
else:
    print()
for num in range(0, 10, 3):
    print(num, end = '')
else:
    print()
```

```
01234
01234
23456789
0369
```

`while` loop

- `for` is known as what's called a "definite iteration" (set number)
- `while` can be used to loop while a condition remains true
- Something in the loop code must be used to move to completion

In [5]:

```
principal = float(input('Principal amount: '))
interest_rate = float(input('Interest rate: ')) / 100
num_years = int(input('Number of years: '))

def calc_interest(principal, interest_rate, num_years):
    year = 1
    while year <= num_years:
        principal *= (1 + interest_rate)
        print(f'{year:>3d}\t${principal:,.2f}')
        year += 1

calc_interest(principal, interest_rate, num_years)
```

```
Principal amount: 1000
Interest rate: 5
Number of years: 5
  1     $1,050.00
  2     $1,102.50
  3     $1,157.62
  4     $1,215.51
  5     $1,276.28
```

Exercise One

Implement a square root function using Newton's method (https://en.wikipedia.org/wiki/Newton's_method):

- Prompt the user for input of a positive float
- Starting with some guess for the square root of the number input by user, we can adjust it based on how close guess^2 is to x, producing a better guess: `guess = guess - (guess² - x) / (2 * guess)`
- Repeating the above makes the guess better and better
- Use a starting guess of 1.0, regardless of the input (it works quite well)
- Repeat the calculation 10 times and print each guess along the way

EXTRA: Use a loop to enforce data validation on the input (verify that it is positive)

Exercise Two

Implement a factorial function (`n! = n * (n - 1) * ... * 1`):

- Prompt the user for input of a non-negative integer
- Using a loop, calculate the factorial of the provided integer (using formula above)
- Output the result

EXTRA: Use a loop to enforce data validation on the input (verify that is non-negative)

Exercise Three

Write a function which implements the Collatz Conjecture (https://en.wikipedia.org/wiki/Collatz_conjecture):

- It should accept an integer >= 1 (return false if < 1)
- If it's even, divide it by 2
- If it's odd, multiply it by 3 and add 1
- Stop when the the result is 1
- Return true for success (i.e. when you reach 1)

- Output the result of each step in sequence and the total count of steps taken to get to 1