Short python tutorial - part 2

Cheatsheet for bitwise operators

`and` : returns True if statement1 and statement2 are True, returns False otherwise
syntax: statement1 `and` statement2

`or` : returns True if statement1 or statement2 is True, returns False otherwise
syntax: statement1 `or` statement3

`not` : returns opposite of statement
syntax: `not` statement

In [5]:
```
# example usage of bitwise operators

tf = True and False
print("The value of True and False is", tf)

tf = True or False
print("The value of True or False is", tf)

tf = not True
print("The value of not True is", tf)

tf = (True and False) or (True or False)
print("The value of (True and False) or (True or False) is", tf)

tf = not tf
print("The value of not ( (True and False) or (True or False) ) is", tf)
```

```
The value of True and False is False
The value of True or False is True
The value of not True is False
The value of (True and False) or (True or False) is True
The value of not ( (True and False) or (True or False) ) is False
```

Flow control

flow control - things in coding which individual components of a program will run and when.

Ex: if statements, loops, ...

In [11]:
```
# if statement - if some testExpression is Ture, evaluate the statements
# inside of the if-statement. Indicate that some code is inside an if-statement
# by indentation

# syntax:        if testExpression:
#                    statements

tf = False
if tf == False:
    print("Hello, World!")

if not tf:
    print("Hello, World!")

num = 10
if num > 4:
    print("The number is greater than 4")
```

```
Hello, World!
Hello, World!
The number is greater than 4
```

In [12]:
```
# if statements can include other conditions and things to do if the first
# statement isn't true. This is called an "if-else-statement"

# syntax:        if textExpression1:
#                    statements1
#                elif textExpression2:
#                    statements2
#                else:
#                    statements3

# if-else-statements will execute statements based on the first True
# textExpressions or will execute statements in the else-statement if none of
# the expressions are True.

# note: the "else" part cannot have any testEpressions
# note: you can have a many else-if's as you want but you can only have
# 1 or 0 else's.

num = 1
if num < 0:
    print("the number is less than 0")
elif num < 3:
    print("the number is between 0 and 2")
else:
    print("the number is greater than 2")
```

```
the number is between 0 and 2
```

In [20]:
```
# For loop - used to iterate over a sequence of things

# syntax:         for val in sequence:
#                     statements

# "val" takes on every value in the sequence and execute "statements" with
# that value.


# program to find the sum of all numbers stored in a list
numbers = [6, 5, 7, 0 , 2, -1]
s = 0

for val in numbers:
    s = s + val

#print(s)



# Program to execute code n times
n = 10
for i in range(3, n):
    print(i)
```

```
3
4
5
6
7
8
9
```

In [29]:
```
# while loop- repeat a specific block of code as long as the test expression
# is true

# syntax:      while testExpression:
#                  statements


# Program to add natural numbers up to n: sum = 1 + 2 + ... + n

n = 10
s = 0
i = 1

while i <= n:
    s = s + i
    i = i + 1

print("The sum is:", s)



# Program a while loop that multiplies all natural numbers up to 7
n = 3
prod = 1
i = 1

while i <= n:
    prod = prod * i
    i = i + 1

print(ans)
```

```
The sum is: 55
6
```

In [ ]:
```
# Break and continue - keywords used to alter the flow of a normal loop
# When break is evaluated, if "breaks" out of the loop containing it
# When continue is evaluated, it skips the rest of the code inside the loop
#   for the current iteration only
```

 Drawing Drawing

In [31]:
```
# Break and continue examples

# for val in "string":
#     if val == "i":
#         break
#     print(val)

for val in "string":
    if val == "i":
        continue
    print(val)
```

```
s
t
r
n
g
```

In [35]:
```
# Python function

# syntax:             def function_name(parameters):
#                         """Description of the function"""
#                         statements

def greet(name):
    """ Function the greets the person named 'name' """
    print("Hello, " + name + ". Good Morning!")

greet("Cindy")
```

```
Hello, Cindy. Good Morning!
```

In [41]:
```
# return keyword is used to exit a function. It also allows you to use wahtever you calculated inside
# the function outside of the function

def abs_val(num):
    """ Computing the absolute value of input argument 'num' """
    if num >= 0:
        return num
    else:
        return -num

a = -10
a_abs = abs_val(a)
print(a_abs)
```

```
10
```

In [43]:
```
# matrices - using lists (actually a list of lists)
L = [[1, 2], [3, 4]]

D = [[0, 1, 2, 3],[1, 2, 3, 4], [2, 3, 4, 5], [3, 4, 5, 6]]
print(D)
```

```
[[0, 1, 2, 3], [1, 2, 3, 4], [2, 3, 4, 5], [3, 4, 5, 6]]
```

In [44]:
```
# matrices - using numpy, a more efficient way to work with matrices
# numpy is a python library with a bunch of useful matrix and linear algebra function

import numpy

C = numpy.array(D)
print(C)
```

```
[[0 1 2 3]
 [1 2 3 4]
 [2 3 4 5]
 [3 4 5 6]]
```

In [47]:
```
n = 4

D_another = numpy.zeros((n, n))
print(D_another)
for i in range(n):
    for j in range(n):
        D_another[i][j] = i + j

print(D_another)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
[[0. 1. 2. 3.]
 [1. 2. 3. 4.]
 [2. 3. 4. 5.]
 [3. 4. 5. 6.]]
```

In [49]:
```
# relevant numpy functions

print(numpy.shape(C))
print(numpy.shape(C)[0])
```

```
(4, 4)
4
```

In [ ]:
```
```