😊 Some useful Operators in Python

1. There are a few built-in functions and "operators" in Python that don't fit well into any category, so we will try to cover them in this Lab.
2. The **range function** allows you to quickly *generate* a list of integers. This comes in handy a lot, so take note of how to use it! There are 3 parameters you can pass: a start, a stop, and a step size.

```
[1]: range(0,11)
```

```
[1]: range(0, 11)
```

3. Note that this is a **generator** function, so to actually get a list out of it, we need to cast it to a list with **list()**. What is a generator? It's a special type of function that will generate information and not need to save it to memory.

```
[3]: # Notice how 11 is not included, up to but not including 11, just like slice notation!
     list(range(0,11))
```

```
[3]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
[4]: list(range(0,12))
```

```
[4]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
[6]: # Third parameter is step size!
     # step size just means how big of a jump/leap/step you
     # take from the starting number to get to the next number.

     list(range(0,11,2))
```

```
[6]: [0, 2, 4, 6, 8, 10]
```

```
[7]: list(range(0,101,10))
```

```
[7]: [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

4. **Enumerate** is a very useful function to use with for loops. Because it's so usual to keep track of how many loops you've completed, enumerate was developed to eliminate the need for you to create and update the index_count or loop_count variables.

[8]:
```
index_count = 0

for letter in 'abcde':
    print("At index {} the letter is {}".format(index_count,letter))
    index_count += 1
```

```
At index 0 the letter is a
At index 1 the letter is b
At index 2 the letter is c
At index 3 the letter is d
At index 4 the letter is e
```

[10]:
```
# Notice the tuple unpacking!

for i,letter in enumerate('abcde'):
    print("At index {} the letter is {}".format(i,letter))
```

```
At index 0 the letter is a
At index 1 the letter is b
At index 2 the letter is c
At index 3 the letter is d
At index 4 the letter is e
```

5. The **zip()** function in Python aggregates elements from multiple iterables (like lists, tuples, or strings) into an iterator of tuples. It pairs elements at the same index from each input iterable, creating tuples where the i-th tuple contains the i-th elements from each iterable. The resulting iterator stops when the shortest input iterable is exhausted.

[12]:
```
list(enumerate('abcde'))
```

[12]:
```
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd'), (4, 'e')]
```

```
[13]: mylist1 = [1,2,3,4,5]
      mylist2 = ['a','b','c','d','e']

[15]: # This one is also a generator! We will explain this later, but for now let's transform it to a list
      zip(mylist1,mylist2)

[15]: <zip at 0x1d205086f08>

[17]: list(zip(mylist1,mylist2))

[17]: [(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd'), (5, 'e')]
```

To use the generator, we could just use a for loop

```
[20]: for item1, item2 in zip(mylist1,mylist2):
          print('For this tuple, first item was {} and second item was {}'.format(item1,item2))

      For this tuple, first item was 1 and second item was a
      For this tuple, first item was 2 and second item was b
      For this tuple, first item was 3 and second item was c
      For this tuple, first item was 4 and second item was d
      For this tuple, first item was 5 and second item was e
```

6. The **in operator** in Python is a membership operator that checks if a value exists within a sequence, such as a list, tuple, string, set, or dictionary. It returns True if the value is found and False otherwise.

```
[21]: 'x' in ['x','y','z']

[21]: True

[22]: 'x' in [1,2,3]

[22]: False
```

7. The **not in operator** in Python checks if a value is not present within a sequence (like a list, tuple, or string) and returns True if the value is not found, and False otherwise.

not in

We can combine **in** with a **not** operator, to check if some object or variable is not present in a list.

```
[1]: 'x' not in ['x','y','z']

[1]: False

[2]: 'x' not in [1,2,3]

[2]: True
```

8. In Python, the **min() and max()** functions are built-in functions used to determine the smallest and largest elements, respectively, from an iterable (like a list, tuple, or string) or a set of arguments.

9. Let’s check the minimum and maximum of the list as you can see below in the snapshot.

```
[26]: mylist = [10,20,30,40,100]

[27]: min(mylist)

[27]: 10

[44]: max(mylist)

[44]: 100
```

10. Python comes with a built-in random library. There are a lot of functions included in this random library, so we will only show you two useful functions for now.

```
[29]: from random import shuffle
      ...

[35]: # This shuffles the list "in-place" meaning it won't return
      # anything, instead it will effect the list passed
      shuffle(mylist)
      ...

[36]: mylist

[36]: [40, 10, 100, 30, 20]

[39]: from random import randint
      ...

[41]: # Return random integer in range [a, b], including both end points.
      randint(0,100)

[41]: 25

[42]: # Return random integer in range [a, b], including both end points.
      randint(0,100)

[42]: 91
```

11. The **input()** function in Python is used to get user input from the console. When **input()** is called, the program's execution pauses, allowing the user to type in a value. Once the user presses Enter, the function returns the entered text as a string.

[43]:
```
input('Enter Something into this box: ')
```

```
Enter Something into this box: great job!
```

[43]:
```
'great job!'
```