Universidad Autónoma de San Luis Potosí

# Python Basics
# Day 2

# Python Casting

Casting in python is therefore done using constructor functions:

- `int()` - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)

- `float()` - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)

- `str()` - constructs a string from a wide variety of data types, including strings, integer literals and float literals

# More about Strings data type

- You can assign a multiline string to a variable by using three quotes.
- Python has a set of built-in methods that you can use on strings.
- To concatenate, or combine, two strings you can use the + operator.
- Using f-strings for formatting.
- Escape character \ (avoid errors using illegal characters inside a string)

# If/elif/else - Conditional Statement

Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

`If – elif`

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

# If/elif/else - Conditional Statement

**Short Hand if**
If you have only one statement to execute, you can put it on the same line as the if statement.

```
if a > b: print("a is greater than b")
```

**Short Hand if – else (ternary operator)**
If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

- **Nested if**
- **Pass statement**

# Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

# While Loops

With the `while` loop we can execute a set of statements as long as a condition is true.

```
i = 1
while i < 6:
    print(i)
    i += 1
```

**Note:** remember to increment i, or else the loop will continue forever.

With the `break` statement we can stop the loop even if the while condition is true.

With the `continue` statement we can stop the current iteration, and continue with the next.

With the `else` statement we can run a block of code once when the condition no longer is true.

# For Loops

A `for` loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

```
for counter in range(1,11):
    print(counter)
```

The `for` loop does not require an indexing variable to set beforehand.

Looping trough a string.

With the `break` statement we can stop the loop before it has looped through all the items.

With the `continue` statement we can stop the current iteration of the loop, and continue with the next.