Week 2: Tuesday

Python Control Flow

Boolean Logic

Operator	Description
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

If, Else, and Elif

```
if condition:
    # code block
elif condition:
    # code block
elif condition:
    # code block
else:
    # code block
```

If Statements, cont.

```
condition = {"a": True, "b": True}
if condition["a"]:
    print("a is True")
```

```
elif not condition["b"]:
    print("b is False")
else:
    print("neither a nor b is True")
```

```
a is True
```

If Statements, cont.

```
condition = {"a": True, "b": True}
if condition["a"]:
    print("a is True")
if condition["b"]:
    print("b is True")
if not condition["b"] and not condition["a"]:
    print("neither a nor b is True")
```

```
a is True
b is True
```

Logical Operators

Operator	Description
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Important Resources

- The best resource is always the Python Documentation.
- Full explanation of Python Control Flow at the above address.

Booleans: true/false values

- Booleans are True/False values.
- They are used to make decisions in code.
- They are often the result of a comparison.

Boolean Example

```
x = 7

my_boolean = x >= 5

print("is x greater or or equal to 5? ", my_boolean)
```

```
is x greater or or equal to 5?  True
```

- `my_boolean` evaluates to `True` if `x` is greater than or equal to 5.
- In other words, python stores `my_boolean` as a `True` or `False` value
  - not as `x >= 5`

if Statements

- `if` statements are used to make decisions in code.
- They are followed by a boolean expression.
- If the boolean expression is `True`, the code block is executed.
- If the boolean expression is `False`, the code block is skipped.

if Statement Example

```
x = 7

my_boolean = x >= 5

if my_boolean:
    print("of course x is greater than 5")
else:
    print("No, stupid, x is less than 5")
```

```
of course x is greater than 5
```

Truthy vs Falsy Values

- The important point is that the value to the right will only be read if the value to the left evaluates to truthy
- Falsy is any value that evaluates to `False`
  - An empty list `[]`, tuple `()`, set `set()`, string `""`, or dictionary `{}`
  - Any Numeric zero: `0`, `0.0`, or complex `3j`
  - Any constant: `False` (obviously), or `None`
- Truthy = everything else.

Truthy / Falsy Examples

```
a = []
b = 3
c = 0
d = 2 - 2

if a:
    print("a is truthy")
else:
    print("a is falsy")

if b:
    print("b is truthy")
else:
    print("b is falsy")
print("d equals ", d)
print("so d evaluates to: ", bool(d))
```

```
a is falsy
b is truthy
d equals  0
so d evaluates to:  False
```

Operators: and

- The and operator returns if both operands are `True`.
- If the first operand is `False`, the second operand is not evaluated.

Operators: and, example 1

```
x = True
y = False
z = True

if x and y:
    print("This will not print because both x and y are not true")
else:
    print("box x and y are not true")

if x and z:
    print("But both x and z evaluate to true, so you see me")
elif z and x:
    print("This will not print because the first if is true")
else:
    print("box x and y are not true")

if x and z:
    print("But both x and z evaluate to true, so you see me")
else:
    print("You will not see this, because x and z evaluate to true")
```

```
box x and y are not true
But both x and z evaluate to true, so you see me
But both x and z evaluate to true, so you see me
```

Operators: or

- The `or` operator evaluates to `True` if either of the operands is `True`.

```
x = True
y = False
z = True
a = 1
b = 0

if x or y:
    print("This first if will print because x is true")
else:
```

```
    print("box x and y are not true")

if a or b == 0:
    print("This will print because b == 0 is true")
else:
    print("You will not see this, because b is true")
```

```
This first if will print because x is true
This will print because b == 0 is true
```

Operators: not

- The `not` operator negates the boolean value of the operand.

```
x = 10
y = 20
z = 10
a = "Something"
b = True
if not x == y:
    print("x is not equal to y")
else:
    print("x is equal to y")  # this will not print
if type(a) != int:
    print("a is not an integer")
else:
    print("a is an integer")  # this will not print
print(not b)
```

```
x is not equal to y
a is not an integer
False
```

The other not

- != is a comparison operator that means "not equal to"

```
x = 10
y = 20

if x != y:
    print("x is not equal to y")
else:
    print("x is equal to y")  # this will not print
```

```
x is not equal to y
```

Operators: in

- The `in` operator checks if a value is in a sequence.
  - A sequence can be a list, tuple, string, or dictionary.

```
my_list = [1, 2, 3, 4, 5]
a = 3
b = 6
if a and b in my_list:
    print("Both a and b are in my_list")
elif a in my_list:
    print("only a is in my_list")
elif b in my_list:
    print("only b is in my list")
else:
    print("neither a nor b are in my_list")
```

```
only a is in my_list
```

Nested If Statements

- You can nest `if` statements inside other `if` statements.

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

if 3 in my_list:
    if 3 in my_list[:5]:
        if 3 in my_list[:3]:
            print("3 is in the first quarter of the list")
```

```
        else:
            print("3 is in the first 3 of the list")
    else:
        print("3 is in the second half of the list")
else:
    print("3 is not in the list")
```

```
3 is in the first quarter of the list
```