IF Statement and Conditions

Basic Usage

Suppose we want to run different code when certain conditions are (or are not) met.

Empty String Example

```
name = ''

print(name)
```

If we print an empty string we get **`None`**. Ideally, we would like to avoid that situation and print something informative instead. We can use an **`if`** statement to check if a condition is True or False and then run specific code depending the the result of the condition. Condition statements MUST evaluate to True or False.

```
name = ''



if name == '':

    print('Unknown Name')
```

Above we use the conditional operator **`==`** to ask if **`name`** is equal to **`''`** (an empty string). Note the colon (**`:`**) at the end of the if statement and the indention of the line of code following it.

Below is the list of comparison operators you can use in an if statement.

Operator	Description	Example
`==`	Equality: Checks if the operands on both sides are the same	`a == a` is True `a == b` is False
`!=`	Not Equal: Check if the operands on both sides are different	`a != b` is True `a != a` is False
`>`	Greater Than: Check if the left-hand side operand is greater than the right-hand side operand	`1 > 0` is True `-1 > 0` is False `1 > 1` is False
`<`	Less Than: Check if the left-hand side operand is less than the right-hand side operand	`0 < 1` is True `0 < -1` is False `1 < 1` is False
`>=`	Greater Than or Equal To: Check if the left-hand side operand is greater than or equal to the right-hand side operand	`1 >= 0` is True `1 >= 1` is True `-1 >= 0` is False
`<=`	Less Than: Check if the left-hand side operand is less than or equal to the right-hand side operand	`0 <= 1` is True `1 <= 1` is True `0 <= -1` is False

(NOTE: There are other types of conditional operations that we will discuss later.)

Back to our code, nothing happens if we actually have a value for name. We'll change that by adding an **`else`** clause.

```
name = 'Kowshik'



if name == '':

   print('Unknown Name')

else:

   print('Hello, ' + name + '!')
```

Now if the first condition is found to be False, then code in the **`else`** clause gets executed.

Let's make this a bit better. We'll ask the user to type in a name.

```
name = input('Please enter a name: ')



if name == '':

   print('Unknown Name')

else:

   print('Hello, ' + name + '!')
```

The **`input()`** function takes a string argument to display a request for data from the user (the string can be empty, but that usually isn't good practice). The user can then type in some text (using Thonny's Shell window) with the keyboard and submit it to the program by pressing the **`enter`** key. The text is then returned by the **`input`** function as a String.

We are not limited to two possible outcomes for an if statement. We can create more conditions before the **`else`** statement using **`elif`** (read as else if).

```
name = input('Please enter a name: ')



if name == '':

   print('Unknown Name')

elif name == 'Kowshik:

   print("Oh...it's you again..")

else:

   print('Hello, ' + name + '!')
```

Here, if the first condition is False we then can check the **`elif`** condition. You can have many **`elif`** statements, but only one **`if`** and **`else`**.

Intermediate Usage

The **`and`** / **`or`** Operators

The if statement is capable of checking conditions with multiple clauses. Assume the simple example which is always **`True`**.

```
if True:

   print('True')

else:

   print('False')
```

Additional conditions can be adding using the logical operators **`and`** or **`or`**.

```
if True or False:
    print('True')
else:
    print('False')
```

The logical operator **`or`** only evaluates to False if both conditions on either side also evaluate False.

```
if True or False:
    print('True')
else:
    print('False')
```

The logical operator **`and`** only evaluates to True if both the conditions on either side are also True.

```
if True and False:
    print('True')
else:
    print('False')
```

The **`or`** logical operator has a higher precedence than **`and`**.

```
if True and False or True:
    print('True')
else:
    print('False')
```

`False or True` is evaluated first, which is True, and then `True and True` also results in True. This is a bit confusing sometimes, so you can always use these truth tables to help you remember.

AND	True	False
True	True	False
False	False	False

OR	True	False
True	True	True
False	True	False

To read these tables, observe where the row and columns intersect. That displays the result of the logical operation indicated in the top left most cell.

The `not` Operator

The `not` operator takes the value of a condition and negates it. In mathematical terms, you can think of the `not` operator like multiplying any number by -1. If the number is positive (True in our case) the result becomes negative (or False). If a number was already negative, then multiplying by -1 makes the number become positive.

This is important because sometimes it is useful to know when something isn't True.

```
account_balance = 10

withdraw_amount = 2



if not (account_balance - withdraw_amount) < 0:

        print('Safe to withdraw funds')

        account_balance = account_balance - withdraw_amount

else:

   print('Account overdrawn')
```

For example, we subtracted the withdraw_amount from the account_balance and then checked if the result of 8 was less than 0. Since 8 < 0 evaluates to False, we need to negate that value in order for the statement to be True and the code to print and subtract the withdraw amount will be run.

The `in` Operator

The **`in`** operator allows you to check if an element exists in a collection. You can also use the **`not in`** operator to check if something isn't present in a collection. A String is an example of a collection as it is made up of a series of characters.

```
my_string = 'apple'



if 'e' in my_string:

   print('Found it!')



if 'z' not in my_string:
```

```
    print('Character not found!')
```

There are other collections that the `in` operator will work with and we'll talk more about that in another guide.

Advanced Usage

None

In python, when variables do not have any data, they are given the value `None`. Variables can be also assigned `None` to indicate that they have no data. A variable assigned an empty string will also have the value `None`.

```
my_value = None

name = ''

print(my_value)

print(name)
```

Running the code above will print the word None out twice.

This can be useful in `if` statements as a variable that holds `None` used in a conditional statement will evaluate to False.

```
name = ''

my_value = None



if name:

    print("There is data!")



if not my_value:

            print("There is no data!")
```

Non Exclusivity

From a logical standpoint, sometimes the code we want to run is based on conditions that are not mutually exclusive (the results of the conditions can impact one another). There is a classic interview question with the following rules:

- Take a number
- If the number is a multiple of three print “Fizz”
- If the number is a multiple of five print “Buzz”
- If the number is both a multiple of three and five print “FizzBuzz”.

Let’s try to implement that and take some user input. Notice that we use the function `int` to convert the String data from the `input` function into an integer. Also recall that the modulus operator (`%`) returns the remainder that results from division. If the remainder is 0, that means the number divides evenly.

```
my_number = int(input("Enter an integer: "))



if (my_number % 3) == 0 and (my_number % 5) == 0:
    print('FizzBuzz')
elif (my_number % 3) == 0:
    print('Fizz')
elif (my_number % 5) == 0:
    print('Buzz')
```

This looks pretty good. However, the condition for printing “FizzBuzz” is directly related to the other two conditions. Could we make the logic easier to understand?

```
my_number = int(input("Enter an integer: "))



output = ''



if (my_number % 3) == 0:

    output += 'Fizz'

if (my_number % 5) == 0:

    output += 'Buzz'



if output:

    print(output)
```

This code exhibits the DRY principle (Don't Repeat Yourself). Instead of checking if our number is divisible by 3 and 5 twice, we now execute each if statement and conditional check once separately and build up a correct answer. The DRY concept is most useful for large projects, but keeping an eye out for redundant code and simplifying when possible is a good habit to build.