Exercise 7: Conditional statements

Conditional statements

If we want to evaulate if something is true or false and then perform different actions depending on the outcome, we can use **`if-else`** statements. An **`if-else`** statement, which is common feature of most programming languages, is a conditional statement: if some condition is true, do something; else, do something different. The else part of the statement is often optional.

Let's assign a number input by the user to a variable, such as `num`.

In [9]:
```
num = input('Enter a number: ')
```

```
Enter a number: 5
```

Now, we will write an if statement to determine if the number falls within a particular range. The Python syntax for an `if else` statement is:

In [ ]:
```
if condition:
    block of code to execute
else:
    block of code to execute
```

Note the colon after the conditional statement and the indentation of code that is to be executed if the condition is met. Indentation is Python's way delimiting blocks of code within conditional statements and in some other contexts. Let us determine if the variable `num` is <10:

In [5]:
```
if num < 10:
    print('The number is less than 10')
else:
    print('The number is greater or equal to 10')
```

```
The number is less than 10
```

You probably got an error message. Look closely at the error message and try to decipher the problem.

By default, the **`input()`** function treats user input as a string, even if it is a number. So we have to specifically tell Python to treat the value as a number. It doesn't matter when we do that as long as it is not after we use the value. To specify a value as an integer, use the syntax **`int(value)`**:

In [10]:
```
if int(num) < 10:
    print('The number is less than 10')
else:
    print('The number is greater or equal to 10')
```

```
The number is less than 10
```

But what if your number has a decimal point?

Recall that numbers containing decimal points belong to a different class called float so we need to specify that the value belongs to the type **`float`** .

Again store a number containing a decimal point as a variable but this time specify that the input should be treated as a floating-point number using **`float()`** :

In [11]:
```
num = float(input("enter a number: "))
```

```
enter a number: 5
```

Now try to use it in an **`if`** statement:

In [12]:
```
if num < 10:
    print('The number is less than 10')
else:
    print('The number is greater or equal to 10')
```

```
The number is less than 10
```

In the code cell below, assign a number input by the user to a variable `num` and using an **`if-else`** statment, identify whether the number is positive or negative and print the results to the screen.

In [15]:
```
num = float(input('Enter a number: '))

if num > 0:
    print('The number is positive')
elif num < 0:
    print('The number is negative')
else:
    print('The number is 0')
```

```
Enter a number: 0
The number is 0
```

Iterations

It is often necessary to repeat an operation multiple times. In Python there are several ways to do this. Here, we'll focus on **`for`** loops

The **`for`** loop allows you to loop over a defined list of objects. Contrast this with the **`while`** loop, which, as saw in bash scripting, which is open ended. **`for`** loops are typically used when we want to repeat a block of code a fixed number times:

`for` loops have the following general structure:

In [ ]:
```
for condition:
    block of code to execute
```

This is a good time to introduce a new function: **`range()`** . The **`range()`** function allows you to specify a range of numbers to iterate through using the following syntax: **`range(start, stop[, step])`** . Essentially, it generates a list of numbers between `start` and `stop` at optional `step` intervals which are generally iterated over in **`for`** loops.

Let's look at a real example:

In [24]:
```
for num in range(1, 11):
    print(num)
```

```
1
2
3
4
5
6
7
8
9
10
```

By default, if you don't specify a starting number, 0 is used:

In [20]:
```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

Let's print every odd number between 1 and 10:

In [22]:
```
for i in range(1,10,2):
    print(i)
```

```
1
3
5
7
9
```

What would range(-10, 10, 2) return?

In [25]:
```
for i in range(-10,10,2):
    print(i)
```

```
-10
-8
-6
-4
-2
0
2
4
6
8
```

Membership and Identify Operators

If we want to determine if something is in a list or string, we can use the **`in`** operator:

In [29]:
```
dna = 'ATGTACGTGCATC'
if 'ATG' in dna:
    print('Start codon')
```

```
Start codon
```

Conversely, we can use the operator **not in** to determine if something is not in a list or string:

In [33]:
```
dna = 'ACGACGCTAGC'
if 'ATG' not in dna:
    print('No start codon')
elif 'ATG' in dna:
    print('Start codon')
else:
    print('Unkown')
```

```
No start codon
```

We can also use **in** to iterate through a string or list:

In [38]:
```
seq = 'ATGATGCTAC'
for nt in seq:
    print(nt)
```

```
A
T
G
A
T
G
C
T
A
C
```

If we want to determine if something has a particular value, we can use the **==** operator:

Let's create a variable sequence that's 3 nt long:

In [39]:
```
seq = 'ATG'
```

Now let's determine if the sequence is a start codon:

In [43]:
```
if seq == 'ATG':
    print('Yes')
```

```
Yes
```

Putting it all together

Let's assign a sequence to a variable and then determine the complement of it using **`if`** statements embedded within a **`for`** loop:

In [47]:
```
seq = 'ATGCTAGCTA'
print('The complement of seq is: ', end='')
for nt in seq:
    if nt == 'A':
        print('T', end='')
    elif nt == 'T':
        print('A', end='')
    elif nt == 'C':
        print('G', end='')
    elif nt == 'G':
        print('C', end='')
```

```
The complement of seq is: TACGATCGAT
```

Next let's cacluate the length of the sequence using the **`len()`** function:

In [45]:
```
print(len(seq))
```

```
10
```

Exercise 7

Write a script that prompts the user for sequence of DNA and converts it to RNA. Recall that input from the command line can be stored as a variable using the **`input()`** function.

!!! Submit your script on Canvas as Exercise 7 !!!

Questions

Q. What is the difference between an `if-else` statement and a `for` loop?

A.

Q. When is indentation used in Python code?

A.

Q. What is the `range()` function?

A.

Q. What is the `in` operator use for?

A.

Practice Problems

Use the table of comparison operators below for the problems.

Operator	Function
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal
!=	not equal

1. Write a script or use the code cell below to prompt the user for a sequence and then calculate the length and print the result.
2. Write a script or use the code cell below to prompt the user for a number and then test whether the number is less than or equal to 10 and print the result.
3. Write a script or use the code cell below to print the numbers 1-100.
4. Write a script or use the code cell below to prompt the user for a sequence and then compute whether the sequence is DNA or RNA. Print the result.
5. Write a script or use the code cell below to prompt the user for a sequence and then compute the numbers of As and Ts it contains. Be sure to print the result.

In [ ]: