Coding Challenge 2025 - Navigators

In the following examples, the symbol **>>** is used to illustrate the output when the preceding line(s) of code is executed.

Input

Getting input

Use the **input** function to get input data. In the coding challenge you must not include any user prompts; just use a simple input statement as shown below.

```python
quantity = input()
```

Some questions will require two lines of input. In this case, use two input prompts.

```python
num1 = input()
num2 = input()
```

Converting between data types (string ↔ integer ↔ float)

Input data is always received as a string of characters. Even if these characters are numbers they must be converted to a numeric data type before they can be processed as numbers. This numeric type can be an integer or a real number (float):

```python
# example 1 - convert input string to an integer

quantity_input = input()
quantity = int(quantity_input)

# example 2 - convert input string to a float (real number)

quantity_input = input()
quantity = float(quantity_input)
```

Output

Use the **print** function to produce output. **Ensure the format is exactly as required.**

Outputting a string or single value

```python
# example 1
print('This is a string')

>> This is a string

# example 2
amount = 5
print(amount)

>> 5
```

Outputting two strings separated by a space

```python
# example 1
first_name = 'Sofia'
last_name = 'Petra'
print(first_name + ' ' + last_name)

>> Sofia Petra

# example 2 (produces the same output as example 1)
print(f'{first_name} {last_name}')

>> Sofia Petra
```

Outputting a sequence of values separated by commas

```python
# example 1

num1 = 23
num2 = 5
num3 = -7
print(str(num1) + ',' + str(num2) + ',' + str(num3))

>> 23,5,-7

# example 2 (produces the same output as example 1)

print(f'{num1},{num2},{num3}')

>> 23,5,-7
```

Strings

Getting the length of a string

```python
my_string = 'crocodile'
my_string_length = len(my_string)
print(my_string_length)

>> 9
```

String concatenation (joining strings)

```python
# example 1

my_string = 'crocodile'
new_string = 'ccc'+'crocodile'
print(new_string)

>> ccccrocodile

# example 2

my_string = 'crocodile'
new_string = 'crocodile' + 'ccc'
print(new_string)

>> crocodileccc
```

Indexing characters in a string

You can access an individual character of a string by specifying its index.

Indexing starts at 0.

```python
my_string = 'crocodile'
first_letter = my_string[0]
print(first_letter)

>> c

my_string = 'crocodile'
sixth_letter = my_string[5]
print(sixth_letter)

>> d
```

String case conversions

```python
# convert to upper case

my_string = 'crocodile'
upper_case_string = my_string.upper()
print(upper_case_string)

>> CROCODILE

# convert to lower case

my_string = 'DESK'
lower_case_string = my_string.lower()
print(lower_case_string)

>> desk
```

Mathematical operators

Examples are shown with the following variables:

a = 7

b = 2

Operator	Meaning	Example	Result
+	addition	a + b	9
-	subtraction	a - b	5
/	division	a / b	3.5
*	multiplication	a * b	14

Rounding

Use the built in function **round** to round to a given number of decimal places.

```
Python
# example - Rounding to 3 decimal places

pi = 3.14159
pi_rounded = round(pi, 3)
print(pi_rounded)

>> 3.142
```

Relational operators

Examples are shown with the following variables:

a = 7

b = 2

Operator	**Meaning**	**Example**	**Result**
==	Equal to	a == b	False
>	Greater than	a > b	True
<	Less than	a < b	False
>=	Greater than or equal to	a >= b	True
<=	Less than or equal to	a <= b	False
!=	Not equal to	a != b	True

Selection

Selection is used to run a block of statements if a condition evaluates as **True**.

```python
temp = 20
if temp < 16:
     print('It is chilly out today')

>>
```

This program will produce no output as the temperature (temp) is >= 16. The condition evaluates as False so the indented statement is skipped.

An else statement provides one or more statements to be executed if the initial condition evaluates as **False**.

```python
temp = 20
if temp < 16:
     print('It is chilly out today')
else:
     print('It is warm out today')

>> It is warm out today
```

You can specify multiple conditions using `elif` statements.

```python
temp = 20
if temp < 16:
     print('It is chilly out today')
elif temp > 27:
     print('It is hot out today')
else:
     print('It is warm out today')

>> It is chilly out today
```

While loops (condition controlled)

A **while loop** is used to specify that an indented block of statements will be executed while the loop condition evaluates as **True**. In the following example the loop condition checks whether a is greater than b:

```Python
a = 7
b = 2

while a > b:
     print (f'{a} is greater than {b}')
     b = b + 1

>> 7 is greater than 2

>> 7 is greater than 3

>> 7 is greater than 4

>> 7 is greater than 5

>> 7 is greater than 6
```

Sometimes you might make a mistake in your code and the while loop condition always evaluates as **True**. This is an infinite loop. You can stop your code running in the Python IDLE by pressing ESC. If you use a different IDE make sure you know how to halt your code.

For loops (count controlled)

If you know how many times you want the indented block of statements code to run, you can use a **for loop**. In the following example, the indented block will be run 3 times (determined by the value 3 in the line `for i in range(3)`).

```
Python
for i in range(3):
     print ('Hello')

>> Hello

>> Hello

>> Hello
```

You can keep the output on a single line by using an end of line character:

```
Python
# example 1 - a blank end of line character

for i in range(3):
     print ('Hello', end = ' ')

>> Hello Hello Hello

# example 2 - a comma as an end of line character

for i in range(3):
     print ('Hello', end = ',')

>> Hello, Hello, Hello,
```

You can use the value of the iterator variable if you need. In the following example the iterator variable is named **`i`**:

```
Python
for i in range(3):
     print(i)

>> 0

>> 1

>> 2
```

Notice that the sequence of values start at 0 and end at 2. The **range** function generates a sequence of values starting at 0 and up to, but not including, the value specified. You can also specify a specific start value and a step value. For example:

```
Python
for i in range (2,5):
     print(i)

>> 2

>> 3

>> 4
```

```
Python
for i in range (3,10,2):
     print(i)

>> 3

>> 5

>> 7

>> 9
```

Python
```
for i in range (10,0,-2):
     print(i)

>> 10

>> 8

>> 6

>> 4

>> 2
```

Notice that the program stops outputting values after 2 because the range excludes the specified end value.

Iterating over a string of characters

There is a special type of FOR loop that allows you to iterate over the characters of a string.

Python
```
motto = 'Be kind'
for character in motto:
     print(character)

>> B

>> e

>>

>> k

>> i

>> n

>> d
```

In this example **`character`** is a variable. On each iteration of the loop its value is the next character in the string. You can name this variable whatever you wish.