Coding Challenge 2025 - Pathfinders

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

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.

Single input value

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

Two or more input lines

Some questions will require two or more lines of input. In this case use as many input prompts as you need:

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

Converting strings to integers or floats

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
```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/decimal
number)
quantity_input = input()
quantity = float(quantity_input)
```

Splitting input lines

You will often need to get a line of input and split it into lists of distinct words or numbers.

```python
# example 1 - input is a list of words separated by spaces

words_in = input()
word_list = words_in.split()

# example 2 - input is a list of words separated by commas

words_in = input()
word_list = words_in.split(',')
```

The results of both of these operations will be a list of words. Sample run:

```python
numbers_in = input()

# assume the input string is: "12, 3, 15, 65"

input_list = numbers_in.split(',')
print(input_list)

>> ['12', '3', '5', '6']
```

Converting a list of strings to a list of numbers

You now have a list of numbers but they are still string values. To manipulate the numbers arithmetically, you will need to convert them to integers or floats.

```python
# an example that should be easy to follow
number_list = []
for item in input_list:
     number_list.append(int(item))

# a shorter equivalent way
number_list = [int(item) for item in input_list]
```

Output

Use the **print** function to produce output. Make sure that 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

# example 3 (produces the same output as examples 1 and 2)

print(first_name, last_name, sep=' ')

>> Sofia Petra
```

Outputting a set 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_string2 = 'crocodile' + 'ccc'
print(new_string)

>> crocodileccc
```

String indexing (joining strings)

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
```

You can index 'backwards' by starting at the end of a string:

```python
my_string = 'crocodile'
last_letter = my_string[-1]
print(last_letter)

>> e
```

String 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

# convert to title case
my_string = 'diane'
title_case_string = my_string.title()
print(title_case_string)

>> Diane
```

Note that this would convert the first letter of every word in a sentence to upper case

Finding and replacing characters within a string

```
Python
my_string = "Green Eggs"
my_string.replace("e", "o")
print(my_string)

>> Groon Eggs
```

Notice that the upper case E was not replaced because the character ‘E’ is not the same as ‘e’.

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
**	exponentiation	a ** b	49
//	Integer (floor) division	a // b	3
%	Modulus (remainder)	a % b	1 (because 7 divided by 2 is 3 remainder 1)

Rounding

```
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

Loops

While loops

A loop where the indented block of statements will be executed while the condition is True. In the following example the 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

If you know how many times you want the indented block of code to run, you can use a for loop. In the following example, the indented block will iterate 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 i if you need to. For example:

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

>> 3

>> 5

>> 7

>> 9
```

Python
```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.

Using a for loop to iterate through a sequence

Sometimes you will have a sequence of values such as a string or a list that you want to iterate through. Python provides a handy mechanism to do this.

```python
user_name = 'Flora'

for character in user_name:
     print(character)

>> F

>> l

>> o

>> r

>> a
```

See the page of lists for an example of iterating through a list.

Selection

Python
```python
num = 7
if num % 2 == 0:
     print('Your number is even')

>>
```
This program will produce no output as the number is odd.

Python
```python
num = 7
if num % 2 == 0:
     print('Your number is even')
else:
     print('Your number is odd')

>> Your number is odd
```

Python
```python
temp = 15
if temp < 16:
     print('It is chilly out today')
elif temp > 27:
     print('It is hot out today')
else:
     print('It is a nice temperature today')

>> It is chilly out today
```

Lists

Defining a list

```
Python
colours = ['Red', 'Green', 'Yellow', 'Purple']
scores = [12, 234, 7]
```

Accessing an element within a list

```
Python
colours = ['Red', 'Green', 'Yellow', 'Purple']
print(colours[1])

>> Green
```

Modifying an element within a list

```
Python
colours = ['Red', 'Green', 'Yellow', 'Purple']
colours[2] = 'Orange'
print(colours)

>> ['Red', 'Green', 'Orange', 'Purple']
```

Iterating through the elements of a list

```
Python
colours = ['Red', 'Green', 'Yellow', 'Purple']
for c in colours:
     print(c)

>> Red

>> Green

>> Orange

>> Purple
```

Getting the length of a list

```
Python
colours = ['Red', 'Green', 'Yellow', 'Purple']
length = len(colours)
print(length)

>> 4
```

Appending an item to a list

```
Python
colours = ['Red', 'Green', 'Yellow', 'Purple']
colours.append('Black')
print(colours)

>> ['Red', 'Green', 'Yellow', 'Purple', 'Black']
```

Removing an item from a list by position

```
Python
colours = ['Red', 'Green', 'Yellow', 'Purple']
del colours[2]
print(colours)

>> ['Red', 'Green', 'Purple']
```

Removing an item from a list by value

```
Python
colours = ['Red', 'Green', 'Yellow', 'Purple']
colours.remove['Green']
print(colours)

>> ['Red', 'Yellow', 'Purple']
```

You will encounter a value error if the item you are trying to remove does not exist.

Checking whether an element is in a list

```
Python
colour_sought = 'Yellow'
colours = ['Red', 'Green', 'Yellow', 'Purple']
colour_sought in colours

>> True
```

Finding the minimum or maximum value in a list

```
Python
# finding the maximum

scores = [12, 3, 15, 87, 2, 31]
highest_score = max(scores)
print(highest_score)

>> 87

# finding the minimum

scores = [12, 2, 15, 87, 2, 31]
lowest_score = min(scores)
print(lowest_score)

>> 2
```

Trapping errors

When an error (exception) occurs, Python will normally stop and generate an error message. These exceptions can be handled using a try statement:

```python
value = 'one'
try:
  num = int(value)
except:
  num = -1
print(num)

>> -1
```

If the value to convert to an integer is not a number (as in this example), the error will be trapped and the program will continue to run.

```python
value = 'one'
try:
  num = int(value)
except:
  print("the value was not a number")
else:
  print("the value was a number")

>> the value was not a number
```