The-Perfect-Crab-Introduction-to-Programming

Basics concept

`def` is a keyword used to define a function

```
def just_return_it(num):
  return num
```

Function 1: just_return_it:

is a function which has:

1. **A name**: `just_return_it` that we can use to call it
2. **A parameter**: `num` that it takes as input. also informally referred to as 'arguments'
3. **A body**: `return num` that processes the input and
4. **Colon**: means the start of new block

just_return_it takes a piece of data as input, and returns it as output.

```
=➔def add_one(num):
  return num + 1
```

Code	What is it?
def	`def` is a keyword that defines a new function
add_one	`add_one` is the **function name**
(num)	`(num)` is the **parameter list**
num	`num` is a **parameter**
:	The `:` symbol indicates the body should start now
return num + 1	`return num + 1` is a **statement**
num + 1	`num + 1` is an **expression**
num	`num` here is a **variable**
+	`+` is an **operator**
1	`1` is a literal number

Function 2: add_one

`add_one` is a function (little machine) that takes a number as an input parameter, adds one to it, and then returns the result.

The expression

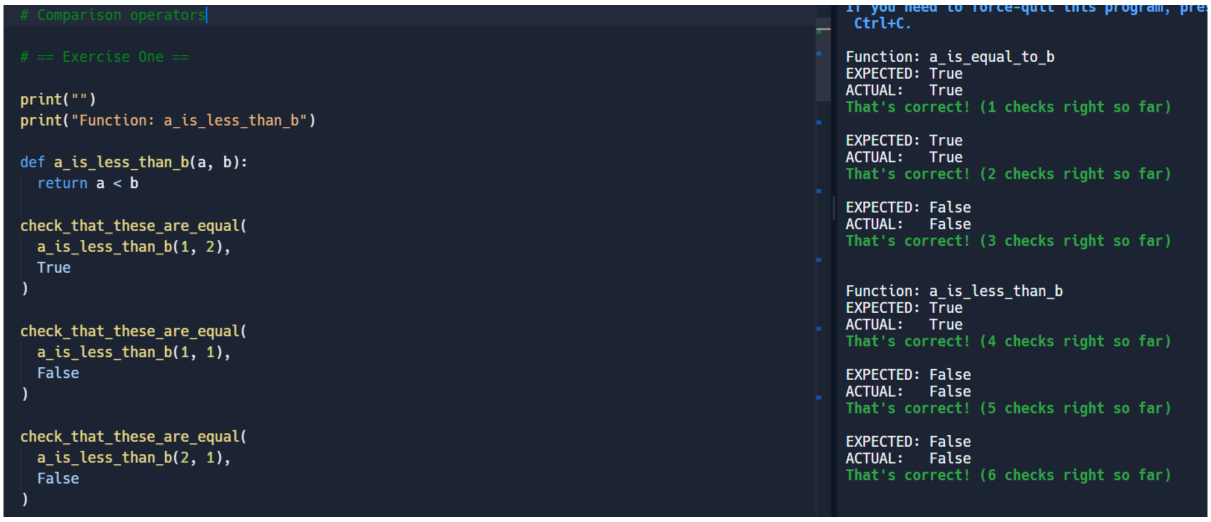
The expression is the fundamental unit of computation in your program. It is the combination of data and operators (and some other things) to produce a result.

Statement

Uses the operators to assign the result. It's called a statement because it changes the 'state' of

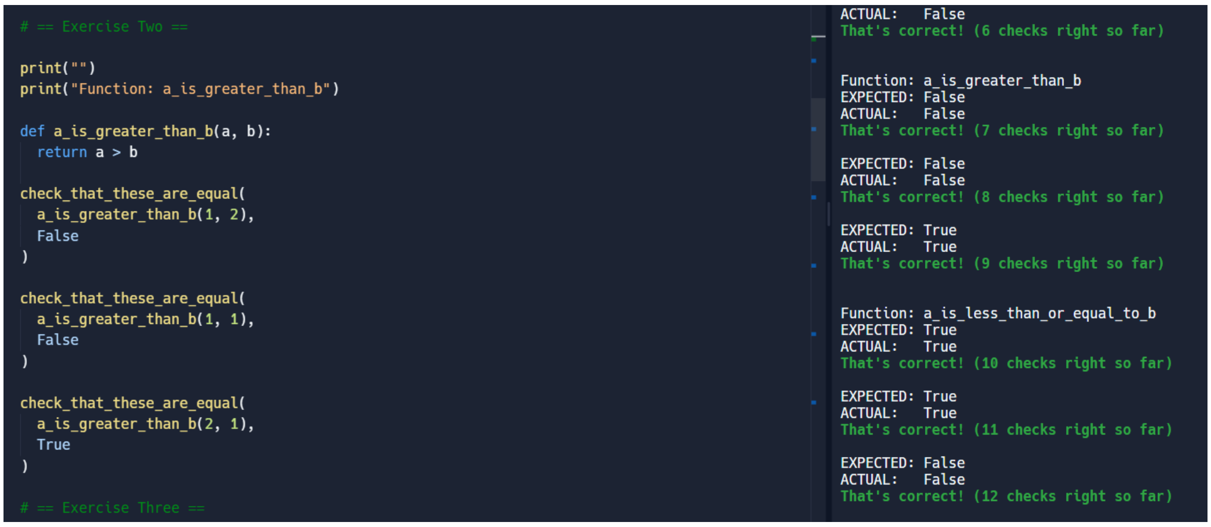
the program.

Comparison operators to evaluate True or False



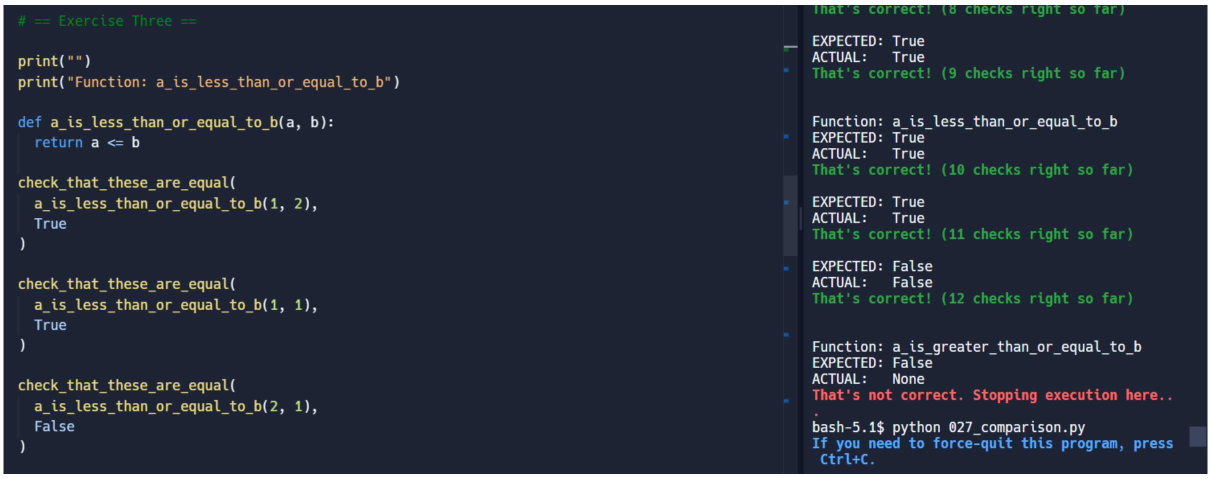
```python
# Comparison operators

# == Exercise One ==

print("")
print("Function: a_is_less_than_b")

def a_is_less_than_b(a, b):
  return a < b

check_that_these_are_equal(
  a_is_less_than_b(1, 2),
  True
)

check_that_these_are_equal(
  a_is_less_than_b(1, 1),
  False
)

check_that_these_are_equal(
  a_is_less_than_b(2, 1),
  False
)
```

```
If you need to force-quit this program, pre
 Ctrl+C.

Function: a_is_equal_to_b
EXPECTED: True
ACTUAL:   True
That's correct! (1 checks right so far)

EXPECTED: True
ACTUAL:   True
That's correct! (2 checks right so far)

EXPECTED: False
ACTUAL:   False
That's correct! (3 checks right so far)


Function: a_is_less_than_b
EXPECTED: True
ACTUAL:   True
That's correct! (4 checks right so far)

EXPECTED: False
ACTUAL:   False
That's correct! (5 checks right so far)

EXPECTED: False
ACTUAL:   False
That's correct! (6 checks right so far)
```



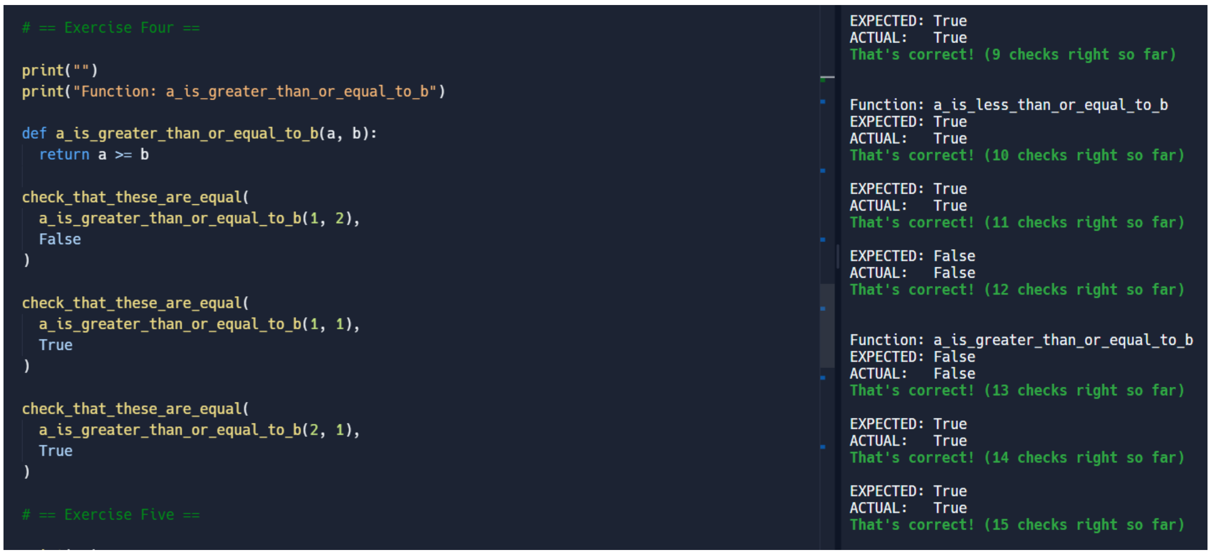
```python
# == Exercise Two ==

print("")
print("Function: a_is_greater_than_b")

def a_is_greater_than_b(a, b):
  return a > b

check_that_these_are_equal(
  a_is_greater_than_b(1, 2),
  False
)

check_that_these_are_equal(
  a_is_greater_than_b(1, 1),
  False
)

check_that_these_are_equal(
  a_is_greater_than_b(2, 1),
  True
)

# == Exercise Three ==
```

```
ACTUAL:   False
That's correct! (6 checks right so far)


Function: a_is_greater_than_b
EXPECTED: False
ACTUAL:   False
That's correct! (7 checks right so far)

EXPECTED: False
ACTUAL:   False
That's correct! (8 checks right so far)

EXPECTED: True
ACTUAL:   True
That's correct! (9 checks right so far)


Function: a_is_less_than_or_equal_to_b
EXPECTED: True
ACTUAL:   True
That's correct! (10 checks right so far)

EXPECTED: True
ACTUAL:   True
That's correct! (11 checks right so far)

EXPECTED: False
ACTUAL:   False
That's correct! (12 checks right so far)
```



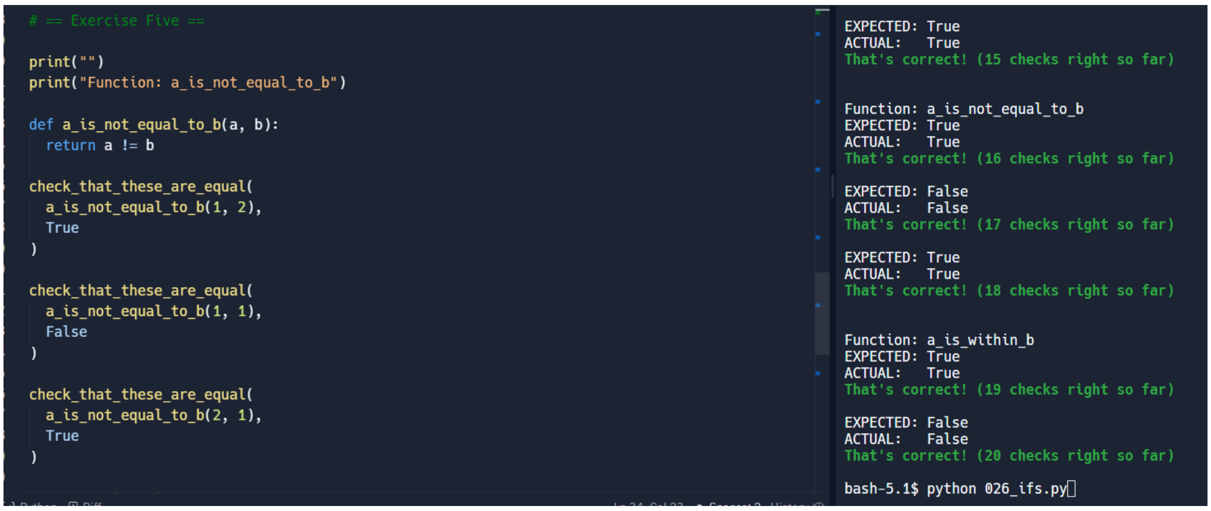
```python
# == Exercise Three ==

print("")
print("Function: a_is_less_than_or_equal_to_b")

def a_is_less_than_or_equal_to_b(a, b):
  return a <= b

check_that_these_are_equal(
  a_is_less_than_or_equal_to_b(1, 2),
  True
)

check_that_these_are_equal(
  a_is_less_than_or_equal_to_b(1, 1),
  True
)

check_that_these_are_equal(
  a_is_less_than_or_equal_to_b(2, 1),
  False
)
```

```
That's correct! (8 checks right so far)

EXPECTED: True
ACTUAL:   True
That's correct! (9 checks right so far)


Function: a_is_less_than_or_equal_to_b
EXPECTED: True
ACTUAL:   True
That's correct! (10 checks right so far)

EXPECTED: True
ACTUAL:   True
That's correct! (11 checks right so far)

EXPECTED: False
ACTUAL:   False
That's correct! (12 checks right so far)


Function: a_is_greater_than_or_equal_to_b
EXPECTED: False
ACTUAL:   None
That's not correct. Stopping execution here..
.
bash-5.1$ python 027_comparison.py
If you need to force-quit this program, press
 Ctrl+C.
```



```
# == Exercise Four ==

print("")
print("Function: a_is_greater_than_or_equal_to_b")

def a_is_greater_than_or_equal_to_b(a, b):
  return a >= b

check_that_these_are_equal(
  a_is_greater_than_or_equal_to_b(1, 2),
  False
)

check_that_these_are_equal(
  a_is_greater_than_or_equal_to_b(1, 1),
  True
)

check_that_these_are_equal(
  a_is_greater_than_or_equal_to_b(2, 1),
  True
)

# == Exercise Five ==
```

```
EXPECTED: True
ACTUAL:   True
That's correct! (9 checks right so far)

Function: a_is_less_than_or_equal_to_b
EXPECTED: True
ACTUAL:   True
That's correct! (10 checks right so far)

EXPECTED: True
ACTUAL:   True
That's correct! (11 checks right so far)

EXPECTED: False
ACTUAL:   False
That's correct! (12 checks right so far)

Function: a_is_greater_than_or_equal_to_b
EXPECTED: False
ACTUAL:   False
That's correct! (13 checks right so far)

EXPECTED: True
ACTUAL:   True
That's correct! (14 checks right so far)

EXPECTED: True
ACTUAL:   True
That's correct! (15 checks right so far)
```



```
# == Exercise Five ==

print("")
print("Function: a_is_not_equal_to_b")

def a_is_not_equal_to_b(a, b):
  return a != b

check_that_these_are_equal(
  a_is_not_equal_to_b(1, 2),
  True
)

check_that_these_are_equal(
  a_is_not_equal_to_b(1, 1),
  False
)

check_that_these_are_equal(
  a_is_not_equal_to_b(2, 1),
  True
)
```

```
EXPECTED: True
ACTUAL:   True
That's correct! (15 checks right so far)

Function: a_is_not_equal_to_b
EXPECTED: True
ACTUAL:   True
That's correct! (16 checks right so far)

EXPECTED: False
ACTUAL:   False
That's correct! (17 checks right so far)

EXPECTED: True
ACTUAL:   True
That's correct! (18 checks right so far)

Function: a_is_within_b
EXPECTED: True
ACTUAL:   True
That's correct! (19 checks right so far)

EXPECTED: False
ACTUAL:   False
That's correct! (20 checks right so far)

bash-5.1$ python 026_ifs.py
```

Logical operators

Difference between Logical and comparison operators:

-Comparison operators evaluate to True or False

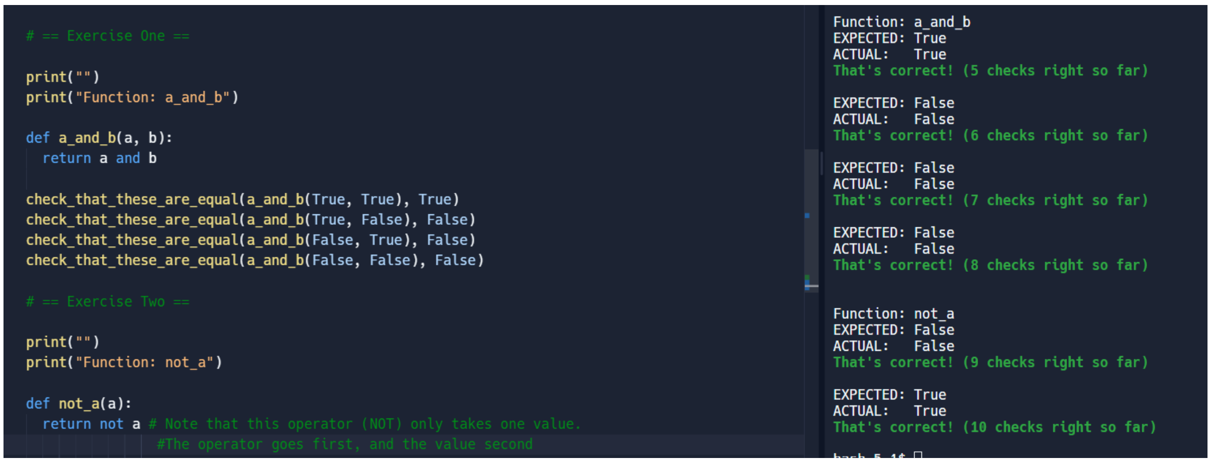
-'logical' or 'Boolean' operators evaluate to true if the condition a condition is met

Logical Operators:

AND (and): Returns True if both statements are true.

OR (or): Returns True if one of the statements is true.

NOT (not): Reverses the result, returns False if the result is true.



```python
# == Exercise One ==

print("")
print("Function: a_and_b")

def a_and_b(a, b):
  return a and b

check_that_these_are_equal(a_and_b(True, True), True)
check_that_these_are_equal(a_and_b(True, False), False)
check_that_these_are_equal(a_and_b(False, True), False)
check_that_these_are_equal(a_and_b(False, False), False)

# == Exercise Two ==

print("")
print("Function: not_a")

def not_a(a):
  return not a # Note that this operator (NOT) only takes one value.
               #The operator goes first, and the value second
```

```
Function: a_and_b
EXPECTED: True
ACTUAL:   True
That's correct! (5 checks right so far)

EXPECTED: False
ACTUAL:   False
That's correct! (6 checks right so far)

EXPECTED: False
ACTUAL:   False
That's correct! (7 checks right so far)

EXPECTED: False
ACTUAL:   False
That's correct! (8 checks right so far)


Function: not_a
EXPECTED: False
ACTUAL:   False
That's correct! (9 checks right so far)

EXPECTED: True
ACTUAL:   True
That's correct! (10 checks right so far)
```

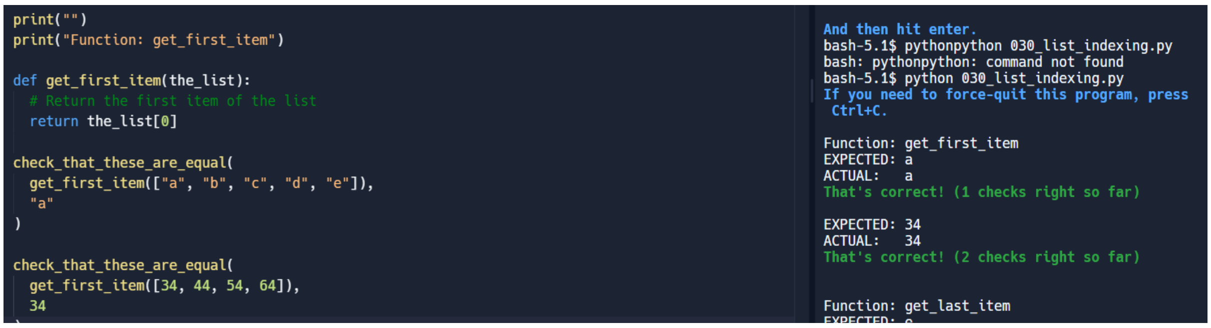
A list and list indexing

- **Definition**: a list is a sequence of items, and those items can be of any type.
- The **square brackets** `[` and `]` tell Python that this is a list, and how the **commas separate** the items in the list.

#Here's are two examples

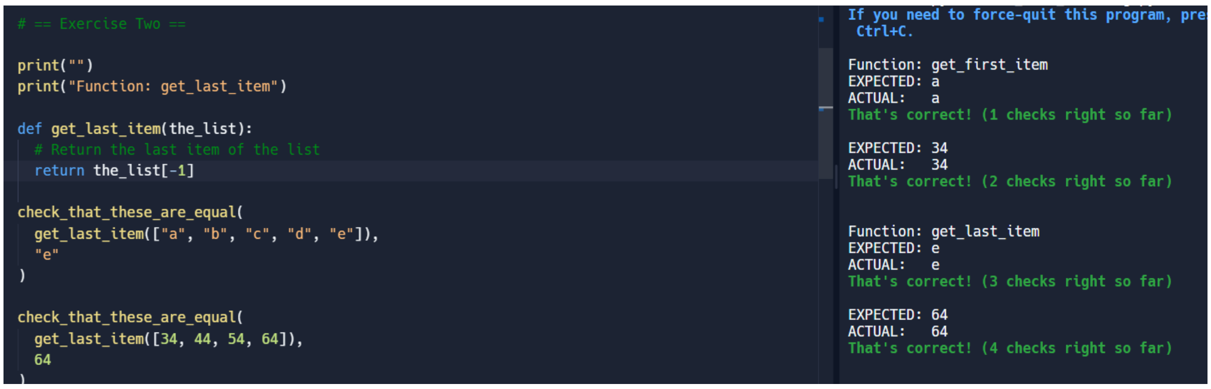
my_favourite_numbers = [1, 3, 5, 7, 9]

my_friends = ["Victoria", "Mel", "Melanie", "Emma"]



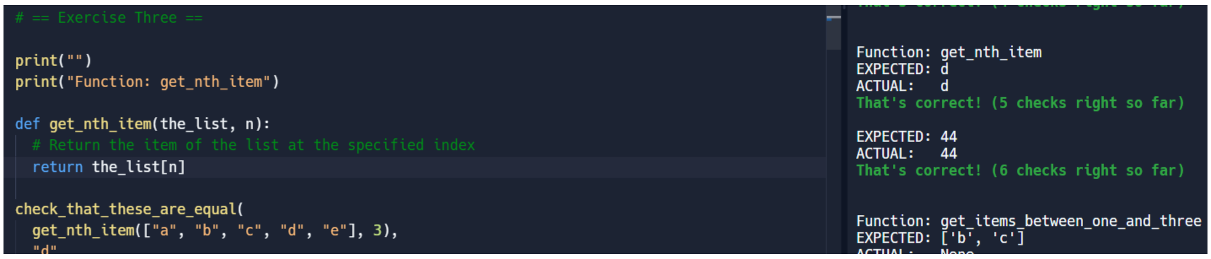
```python
print("")
print("Function: get_first_item")

def get_first_item(the_list):
  # Return the first item of the list
  return the_list[0]

check_that_these_are_equal(
  get_first_item(["a", "b", "c", "d", "e"]),
  "a"
)

check_that_these_are_equal(
  get_first_item([34, 44, 54, 64]),
  34
```

```
And then hit enter.
bash-5.1$ pythonpython 030_list_indexing.py
bash: pythonpython: command not found
bash-5.1$ python 030_list_indexing.py
If you need to force-quit this program, press
 Ctrl+C.

Function: get_first_item
EXPECTED: a
ACTUAL:   a
That's correct! (1 checks right so far)

EXPECTED: 34
ACTUAL:   34
That's correct! (2 checks right so far)


Function: get_last_item
```



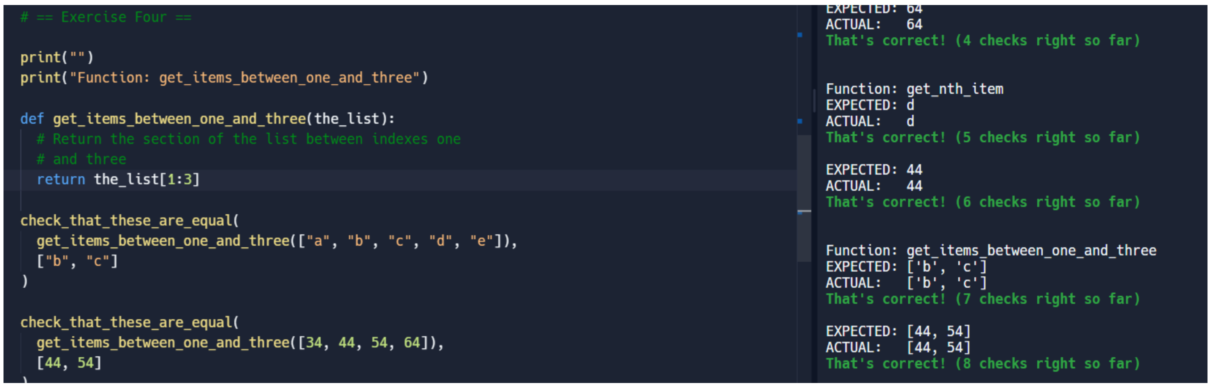
```python
# == Exercise Two ==

print("")
print("Function: get_last_item")

def get_last_item(the_list):
  # Return the last item of the list
  return the_list[-1]

check_that_these_are_equal(
  get_last_item(["a", "b", "c", "d", "e"]),
  "e"
)

check_that_these_are_equal(
  get_last_item([34, 44, 54, 64]),
  64
```

```
If you need to force-quit this program, pre
 Ctrl+C.

Function: get_first_item
EXPECTED: a
ACTUAL:   a
That's correct! (1 checks right so far)

EXPECTED: 34
ACTUAL:   34
That's correct! (2 checks right so far)


Function: get_last_item
EXPECTED: e
ACTUAL:   e
That's correct! (3 checks right so far)

EXPECTED: 64
ACTUAL:   64
That's correct! (4 checks right so far)
```



```
# == Exercise Three ==

print("")
print("Function: get_nth_item")

def get_nth_item(the_list, n):
  # Return the item of the list at the specified index
  return the_list[n]

check_that_these_are_equal(
  get_nth_item(["a", "b", "c", "d", "e"], 3),
  "d"
```

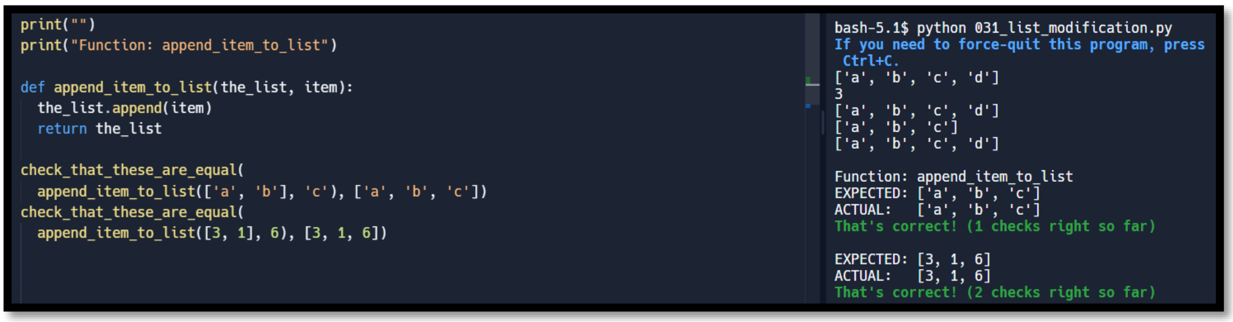
```
Function: get_nth_item
EXPECTED: d
ACTUAL:   d
That's correct! (5 checks right so far)

EXPECTED: 44
ACTUAL:   44
That's correct! (6 checks right so far)

Function: get_items_between_one_and_three
EXPECTED: ['b', 'c']
ACTUAL:   None
```



```
# == Exercise Four ==

print("")
print("Function: get_items_between_one_and_three")

def get_items_between_one_and_three(the_list):
  # Return the section of the list between indexes one
  # and three
  return the_list[1:3]

check_that_these_are_equal(
  get_items_between_one_and_three(["a", "b", "c", "d", "e"]),
  ["b", "c"]
)

check_that_these_are_equal(
  get_items_between_one_and_three([34, 44, 54, 64]),
  [44, 54]
)
```

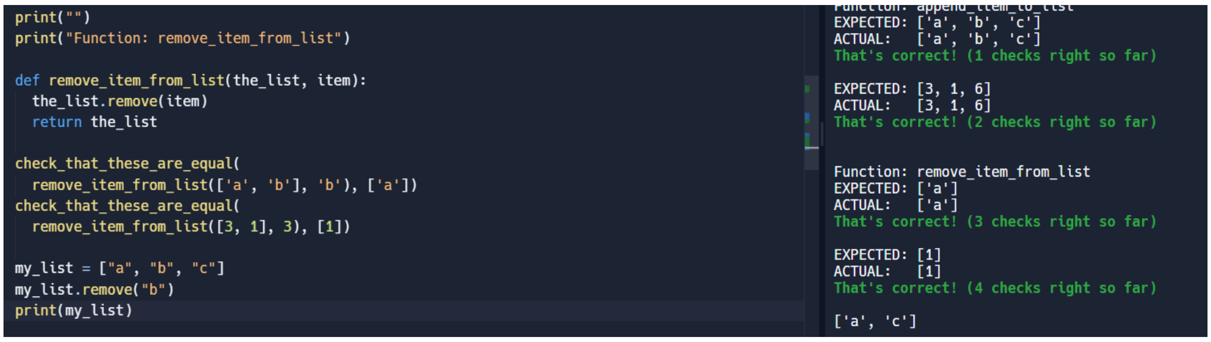
```
EXPECTED: 64
ACTUAL:   64
That's correct! (4 checks right so far)

Function: get_nth_item
EXPECTED: d
ACTUAL:   d
That's correct! (5 checks right so far)

EXPECTED: 44
ACTUAL:   44
That's correct! (6 checks right so far)

Function: get_items_between_one_and_three
EXPECTED: ['b', 'c']
ACTUAL:   ['b', 'c']
That's correct! (7 checks right so far)

EXPECTED: [44, 54]
ACTUAL:   [44, 54]
That's correct! (8 checks right so far)
```

LIST MODIFICATION: append, remove, count, index, length, reverse



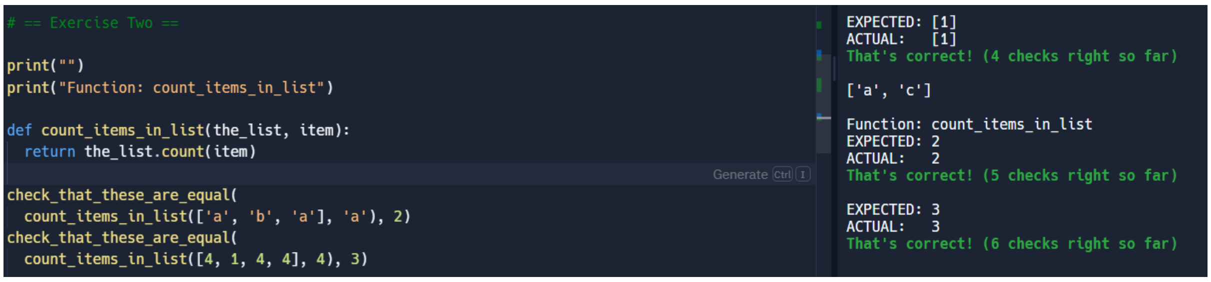
```
print("")
print("Function: append_item_to_list")

def append_item_to_list(the_list, item):
  the_list.append(item)
  return the_list

check_that_these_are_equal(
  append_item_to_list(['a', 'b'], 'c'), ['a', 'b', 'c'])
check_that_these_are_equal(
  append_item_to_list([3, 1], 6), [3, 1, 6])
```

```
bash-5.1$ python 031_list_modification.py
If you need to force-quit this program, press
 Ctrl+C.
['a', 'b', 'c', 'd']
3
['a', 'b', 'c', 'd']
['a', 'b', 'c']
['a', 'b', 'c', 'd']

Function: append_item_to_list
EXPECTED: ['a', 'b', 'c']
ACTUAL:   ['a', 'b', 'c']
That's correct! (1 checks right so far)

EXPECTED: [3, 1, 6]
ACTUAL:   [3, 1, 6]
That's correct! (2 checks right so far)
```



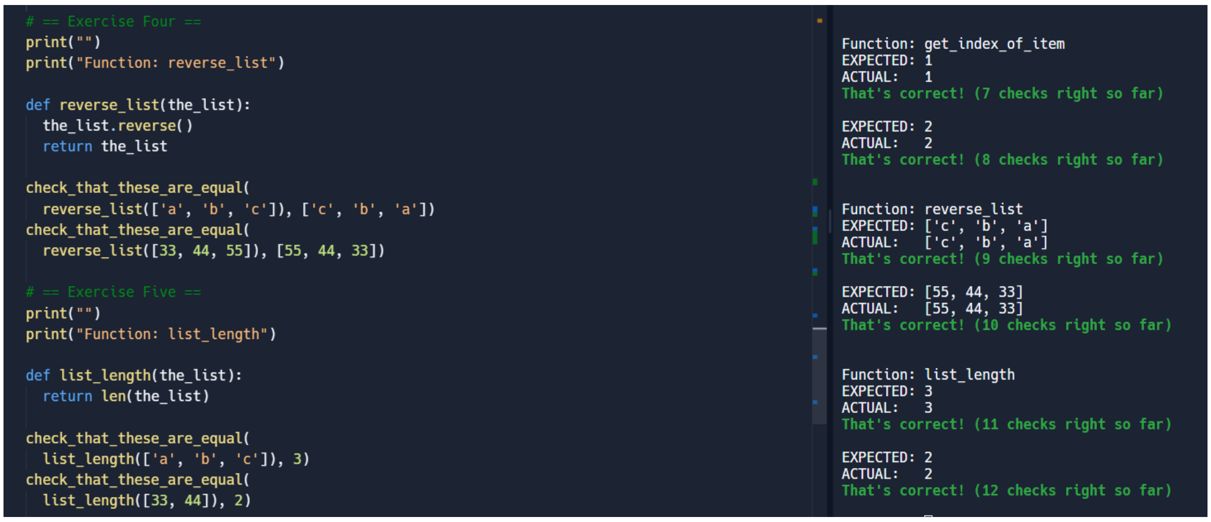
```
print("")
print("Function: remove_item_from_list")

def remove_item_from_list(the_list, item):
  the_list.remove(item)
  return the_list

check_that_these_are_equal(
  remove_item_from_list(['a', 'b'], 'b'), ['a'])
check_that_these_are_equal(
  remove_item_from_list([3, 1], 3), [1])

my_list = ["a", "b", "c"]
my_list.remove("b")
print(my_list)
```

```
EXPECTED: ['a', 'b', 'c']
ACTUAL:   ['a', 'b', 'c']
That's correct! (1 checks right so far)

EXPECTED: [3, 1, 6]
ACTUAL:   [3, 1, 6]
That's correct! (2 checks right so far)

Function: remove_item_from_list
EXPECTED: ['a']
ACTUAL:   ['a']
That's correct! (3 checks right so far)

EXPECTED: [1]
ACTUAL:   [1]
That's correct! (4 checks right so far)

['a', 'c']
```



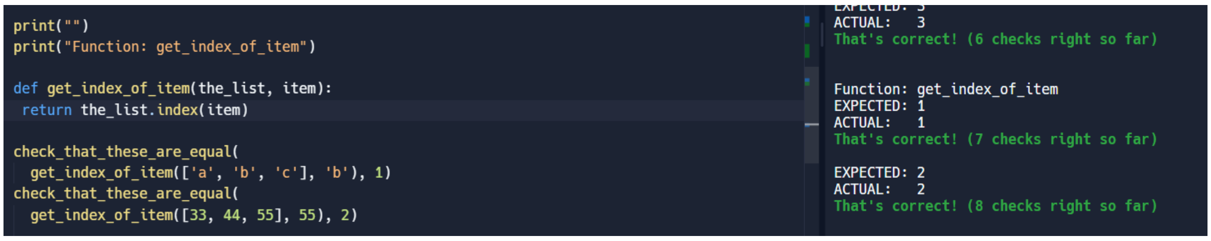
```
# == Exercise Two ==

print("")
print("Function: count_items_in_list")

def count_items_in_list(the_list, item):
  return the_list.count(item)

check_that_these_are_equal(
  count_items_in_list(['a', 'b', 'a'], 'a'), 2)
check_that_these_are_equal(
  count_items_in_list([4, 1, 4, 4], 4), 3)
```

```
EXPECTED: [1]
ACTUAL:   [1]
That's correct! (4 checks right so far)

['a', 'c']

Function: count_items_in_list
EXPECTED: 2
ACTUAL:   2
That's correct! (5 checks right so far)

EXPECTED: 3
ACTUAL:   3
That's correct! (6 checks right so far)
```



```python
# == Exercise Four ==
print("")
print("Function: reverse_list")

def reverse_list(the_list):
  the_list.reverse()
  return the_list

check_that_these_are_equal(
  reverse_list(['a', 'b', 'c']), ['c', 'b', 'a'])
check_that_these_are_equal(
  reverse_list([33, 44, 55]), [55, 44, 33])

# == Exercise Five ==
print("")
print("Function: list_length")

def list_length(the_list):
  return len(the_list)

check_that_these_are_equal(
  list_length(['a', 'b', 'c']), 3)
check_that_these_are_equal(
  list_length([33, 44]), 2)
```

```
Function: get_index_of_item
EXPECTED: 1
ACTUAL:   1
That's correct! (7 checks right so far)

EXPECTED: 2
ACTUAL:   2
That's correct! (8 checks right so far)

Function: reverse_list
EXPECTED: ['c', 'b', 'a']
ACTUAL:   ['c', 'b', 'a']
That's correct! (9 checks right so far)

EXPECTED: [55, 44, 33]
ACTUAL:   [55, 44, 33]
That's correct! (10 checks right so far)

Function: list_length
EXPECTED: 3
ACTUAL:   3
That's correct! (11 checks right so far)

EXPECTED: 2
ACTUAL:   2
That's correct! (12 checks right so far)
```

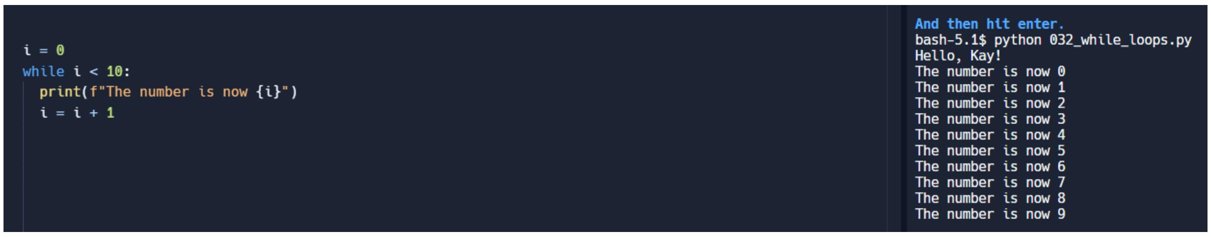


```python
print("")
print("Function: get_index_of_item")

def get_index_of_item(the_list, item):
  return the_list.index(item)

check_that_these_are_equal(
  get_index_of_item(['a', 'b', 'c'], 'b'), 1)
check_that_these_are_equal(
  get_index_of_item([33, 44, 55], 55), 2)
```

```
EXPECTED: 3
ACTUAL:   3
That's correct! (6 checks right so far)

Function: get_index_of_item
EXPECTED: 1
ACTUAL:   1
That's correct! (7 checks right so far)

EXPECTED: 2
ACTUAL:   2
That's correct! (8 checks right so far)
```

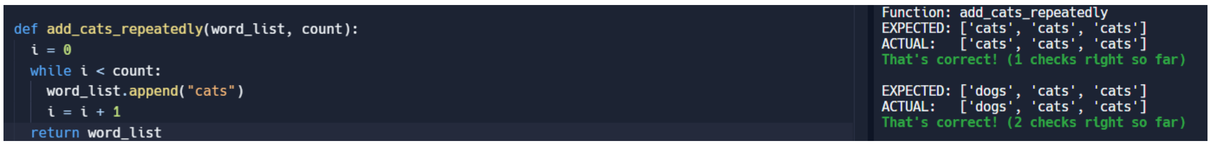
While Loops and For Loop

The **"while" loop** is like an `if`, in that it takes an expression that evaluates to True or False, and then executes its block of code if the condition is True**.**



```python
i = 0
while i < 10:
  print(f"The number is now {i}")
  i = i + 1
```

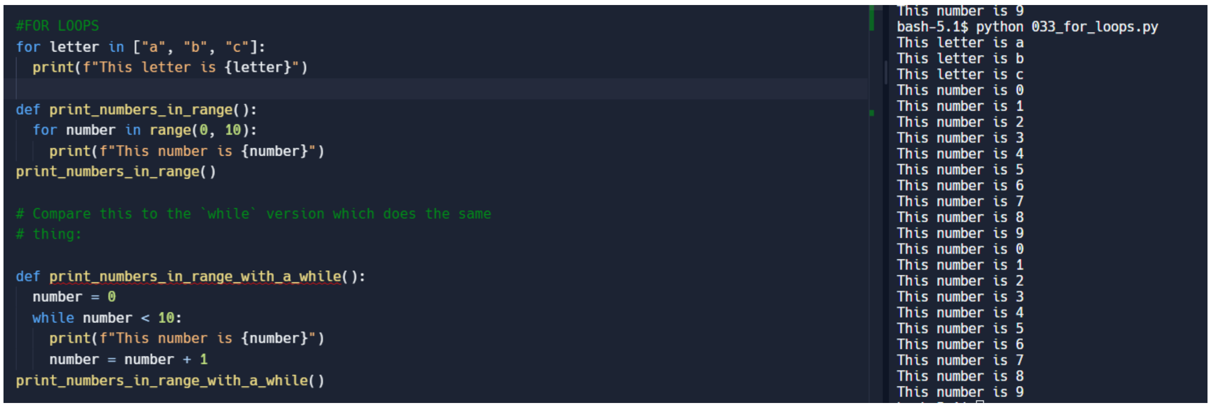
```
And then hit enter.
bash-5.1$ python 032_while_loops.py
Hello, Kay!
The number is now 0
The number is now 1
The number is now 2
The number is now 3
The number is now 4
The number is now 5
The number is now 6
The number is now 7
The number is now 8
The number is now 9
```



```python
def add_cats_repeatedly(word_list, count):
  i = 0
  while i < count:
    word_list.append("cats")
    i = i + 1
  return word_list
```

```
Function: add_cats_repeatedly
EXPECTED: ['cats', 'cats', 'cats']
ACTUAL:   ['cats', 'cats', 'cats']
That's correct! (1 checks right so far)

EXPECTED: ['dogs', 'cats', 'cats']
ACTUAL:   ['dogs', 'cats', 'cats']
That's correct! (2 checks right so far)
```

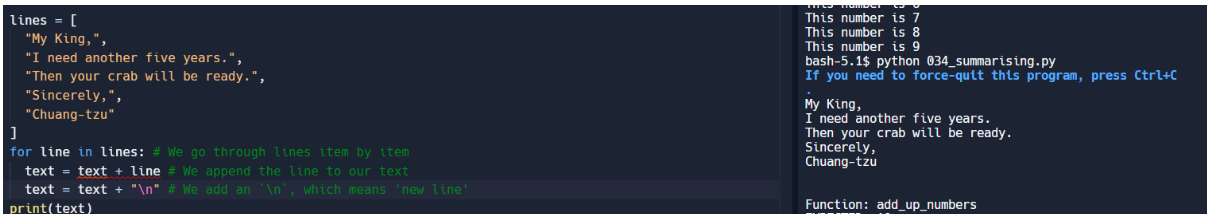
the Python **for loop** takes each item one by one and runs its block of code with that item.



```
#FOR LOOPS
for letter in ["a", "b", "c"]:
  print(f"This letter is {letter}")

def print_numbers_in_range():
  for number in range(0, 10):
    print(f"This number is {number}")
print_numbers_in_range()

# Compare this to the `while` version which does the same
# thing:

def print_numbers_in_range_with_a_while():
  number = 0
  while number < 10:
    print(f"This number is {number}")
    number = number + 1
print_numbers_in_range_with_a_while()
```

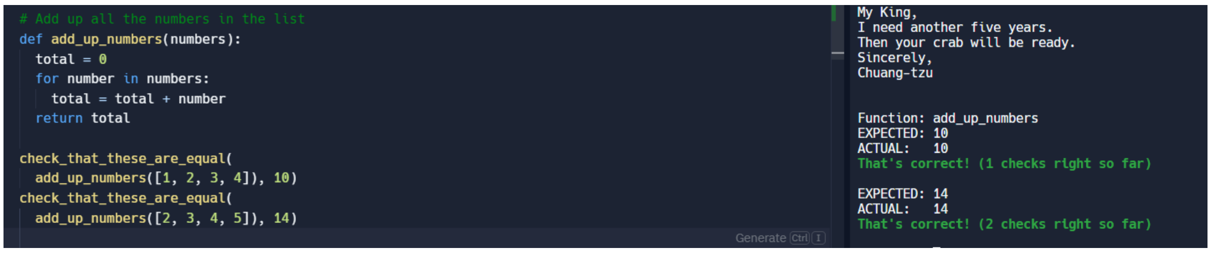
```
This number is 9
bash-5.1$ python 033_for_loops.py
This letter is a
This letter is b
This letter is c
This number is 0
This number is 1
This number is 2
This number is 3
This number is 4
This number is 5
This number is 6
This number is 7
This number is 8
This number is 9
This number is 0
This number is 1
This number is 2
This number is 3
This number is 4
This number is 5
This number is 6
This number is 7
This number is 8
This number is 9
```

Summarising: Using a loop to distil a list into one value.



```
lines = [
  "My King,",
  "I need another five years.",
  "Then your crab will be ready.",
  "Sincerely,",
  "Chuang-tzu"
]
for line in lines: # We go through lines item by item
  text = text + line # We append the line to our text
  text = text + "\n" # We add an `\n`, which means 'new line'
print(text)
```

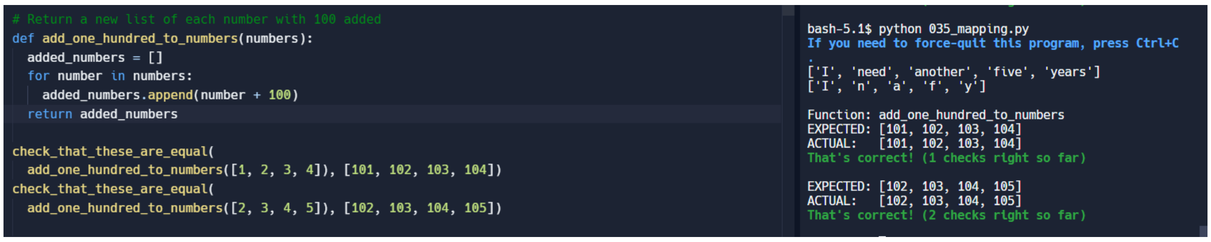
```
This number is 7
This number is 8
This number is 9
bash-5.1$ python 034_summarising.py
If you need to force-quit this program, press Ctrl+C
.
My King,
I need another five years.
Then your crab will be ready.
Sincerely,
Chuang-tzu


Function: add_up_numbers
```



```
# Add up all the numbers in the list
def add_up_numbers(numbers):
  total = 0
  for number in numbers:
    total = total + number
  return total

check_that_these_are_equal(
  add_up_numbers([1, 2, 3, 4]), 10)
check_that_these_are_equal(
  add_up_numbers([2, 3, 4, 5]), 14)
```

```
My King,
I need another five years.
Then your crab will be ready.
Sincerely,
Chuang-tzu


Function: add_up_numbers
EXPECTED: 10
ACTUAL:   10
That's correct! (1 checks right so far)

EXPECTED: 14
ACTUAL:   14
That's correct! (2 checks right so far)
```

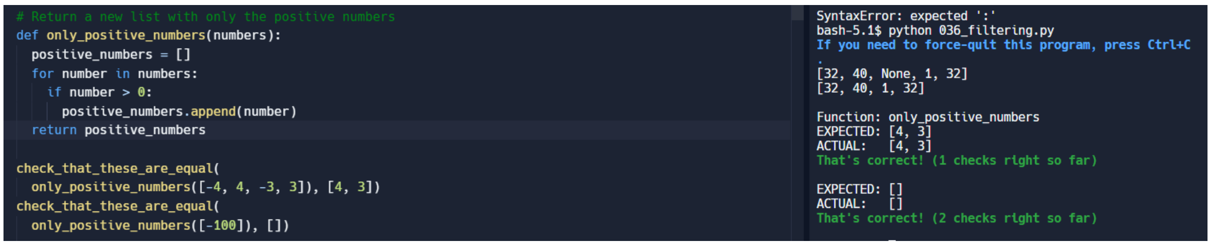
Mapping: Using a loop to convert each item to another item.



```
# Return a new list of each number with 100 added
def add_one_hundred_to_numbers(numbers):
  added_numbers = []
  for number in numbers:
    added_numbers.append(number + 100)
  return added_numbers

check_that_these_are_equal(
  add_one_hundred_to_numbers([1, 2, 3, 4]), [101, 102, 103, 104])
check_that_these_are_equal(
  add_one_hundred_to_numbers([2, 3, 4, 5]), [102, 103, 104, 105])
```

```
bash-5.1$ python 035_mapping.py
If you need to force-quit this program, press Ctrl+C
.
['I', 'need', 'another', 'five', 'years']
['I', 'n', 'a', 'f', 'y']

Function: add_one_hundred_to_numbers
EXPECTED: [101, 102, 103, 104]
ACTUAL:   [101, 102, 103, 104]
That's correct! (1 checks right so far)

EXPECTED: [102, 103, 104, 105]
ACTUAL:   [102, 103, 104, 105]
That's correct! (2 checks right so far)
```

Filtering: Using a loop to pick out only some items from a list.



```
# Return a new list with only the positive numbers
def only_positive_numbers(numbers):
  positive_numbers = []
  for number in numbers:
    if number > 0:
      positive_numbers.append(number)
  return positive_numbers

check_that_these_are_equal(
  only_positive_numbers([-4, 4, -3, 3]), [4, 3])
check_that_these_are_equal(
  only_positive_numbers([-100]), [])
```

```
SyntaxError: expected ':'
bash-5.1$ python 036_filtering.py
If you need to force-quit this program, press Ctrl+C
.
[32, 40, None, 1, 32]
[32, 40, 1, 32]

Function: only_positive_numbers
EXPECTED: [4, 3]
ACTUAL:   [4, 3]
That's correct! (1 checks right so far)

EXPECTED: []
ACTUAL:   []
That's correct! (2 checks right so far)
```

Create a Dictionary

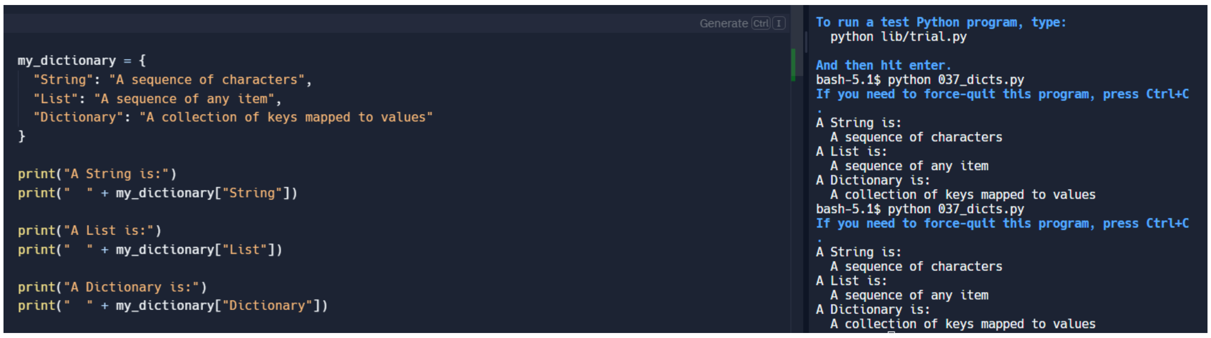
Reminder: "String": "A sequence of characters",

"List": "A sequence of any item",

"Dictionary": "A collection of keys mapped to values"

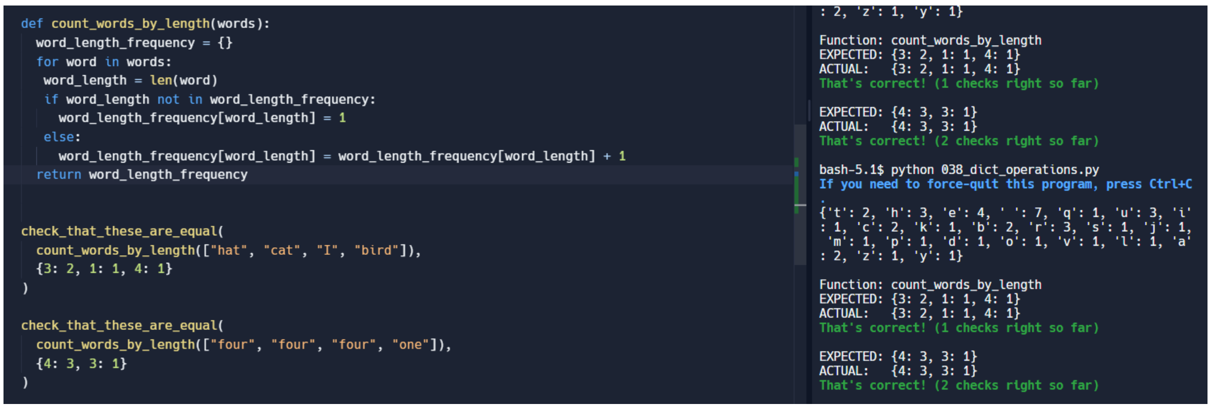
Note:

- In a dictionary you look up a word and it provides a definition?
- In that sense, the *'word' is the key*, and the *'definition' is the value.*
- use braces \`{\` and \`}\` to tell Python that this is a dictionary
- use commas \`,\` to separate pairs
- use colons \`:\` to separate keys and values



```python
my_dictionary = {
  "String": "A sequence of characters",
  "List": "A sequence of any item",
  "Dictionary": "A collection of keys mapped to values"
}

print("A String is:")
print("  " + my_dictionary["String"])

print("A List is:")
print("  " + my_dictionary["List"])

print("A Dictionary is:")
print("  " + my_dictionary["Dictionary"])
```

```
To run a test Python program, type:
  python lib/trial.py

And then hit enter.
bash-5.1$ python 037_dicts.py
If you need to force-quit this program, press Ctrl+C
.
A String is:
  A sequence of characters
A List is:
  A sequence of any item
A Dictionary is:
  A collection of keys mapped to values
bash-5.1$ python 037_dicts.py
If you need to force-quit this program, press Ctrl+C
.
A String is:
  A sequence of characters
A List is:
  A sequence of any item
A Dictionary is:
  A collection of keys mapped to values
```



```python
def count_words_by_length(words):
  word_length_frequency = {}
  for word in words:
   word_length = len(word)
   if word_length not in word_length_frequency:
     word_length_frequency[word_length] = 1
   else:
     word_length_frequency[word_length] = word_length_frequency[word_length] + 1
  return word_length_frequency


check_that_these_are_equal(
  count_words_by_length(["hat", "cat", "I", "bird"]),
  {3: 2, 1: 1, 4: 1}
)

check_that_these_are_equal(
  count_words_by_length(["four", "four", "four", "one"]),
  {4: 3, 3: 1}
)
```

```
: 2, 'z': 1, 'y': 1}

Function: count_words_by_length
EXPECTED: {3: 2, 1: 1, 4: 1}
ACTUAL:   {3: 2, 1: 1, 4: 1}
That's correct! (1 checks right so far)

EXPECTED: {4: 3, 3: 1}
ACTUAL:   {4: 3, 3: 1}
That's correct! (2 checks right so far)

bash-5.1$ python 038_dict_operations.py
If you need to force-quit this program, press Ctrl+C
.
{'t': 2, 'h': 3, 'e': 4, ' ': 7, 'q': 1, 'u': 3, 'i'
: 1, 'c': 2, 'k': 1, 'b': 2, 'r': 3, 's': 1, 'j': 1,
 'm': 1, 'p': 1, 'd': 1, 'o': 1, 'v': 1, 'l': 1, 'a'
: 2, 'z': 1, 'y': 1}

Function: count_words_by_length
EXPECTED: {3: 2, 1: 1, 4: 1}
ACTUAL:   {3: 2, 1: 1, 4: 1}
That's correct! (1 checks right so far)

EXPECTED: {4: 3, 3: 1}
ACTUAL:   {4: 3, 3: 1}
That's correct! (2 checks right so far)
```