Built-in Data Structures I: Strings and Lists

Contents

Strings

- Create strings
  - Enclose characters in either single or double quotation marks

```
this = 'Hello'       # Create a string by single quotes
that = "World"       # Create a string by double quotes

print(this)
print(that)
```

```
Hello
World
```

Strings

- Create strings
  - Create multi-line strings with three single/double quotation marks

```
shining = """
All work and no play makes Jack a dull boy
All work and no play makes Jack a dull boy
         All work and no play
         makes Jack a dull boy
         All work and no play
         makes Jack a dull boy
All work and no play makes Jack a dull boy
All work and no play makes Jack a dull boy
"""
```

Strings

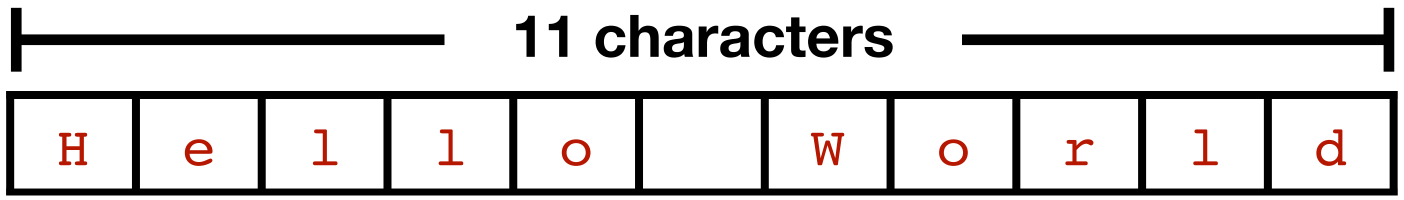
- Create strings
    - Other approaches
        - ✓ The output of the `input()` function
        - ✓ Convert objects of other types to string by the `str()` function
        - ✓ Concatenate or duplicate other strings

Strings

- Sequence operations
  - Length of a string: the `len()` function.

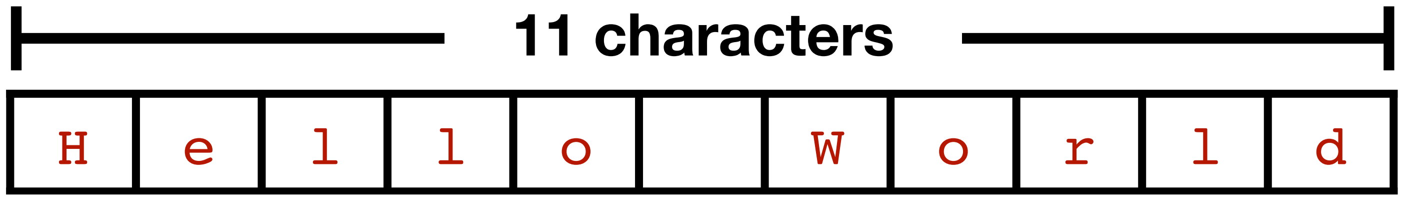
```
greetings = "Hello World"

print(len(greetings))
```

```
11
```



Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing individual characters



Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing individual characters

```
greetings = "Hello World"

letter_e = greetings[1]       # Access the second item via index 1
print(letter_e)

letter_r = greetings[8]       # Access the ninth item via index 8
print(letter_r)
```

Square brackets for indexing

```
e
r
```

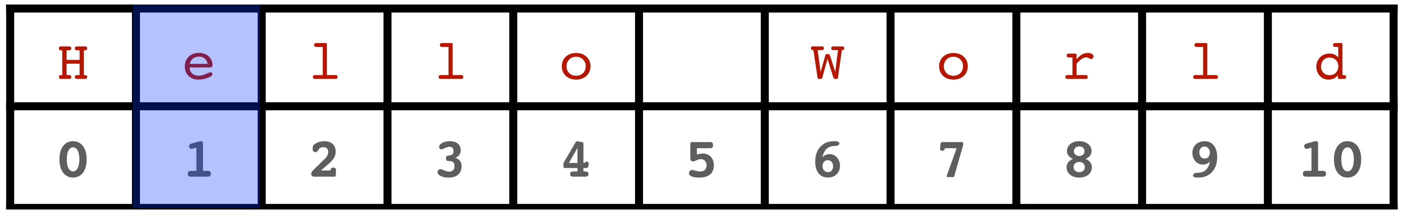
	H	e	l	l	o		W	o	r	l	d
Indexes	0	1	2	3	4	5	6	7	8	9	10

Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing individual characters

```
greetings = "Hello World"

letter_e = greetings[1]      # Access the second item via index 1
print(letter_e)

letter_r = greetings[8]      # Access the ninth item via index 8
print(letter_r)
```

```
e
r
```



H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing individual characters

```
greetings = "Hello World"

letter_e = greetings[1]     # Access the second item via index 1
print(letter_e)

letter_r = greetings[8]     # Access the ninth item via index 8
print(letter_r)
```

```
e
r
```

H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing individual characters

```
greetings = "Hello World"

letter_d = greetings[-1]    # Access the last item
print(letter_d)

letter_l = greetings[-2]    # Access the second to last item
print(letter_l)
```

```
d
l
```

Indexes from the rear

H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing individual characters

```
greetings = "Hello World"

letter_d = greetings[-1]     # Access the last item
print(letter_d)

letter_l = greetings[-2]     # Access the second to last item
print(letter_l)
```

```
d
l
```

H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing individual characters

```
greetings = "Hello World"

letter_d = greetings[-1]     # Access the last item
print(letter_d)

letter_l = greetings[-2]     # Access the second to last item
print(letter_l)
```

```
d
l
```

H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing subsets of strings

H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing subsets of strings

```
[start:stop:step]
```

Arguments	Remarks	Default Values
`start`	The first index of the slice	0
`stop`	The index before which the slice stops	length of the string
`step`	The step length of the slice	1

Strings

- Sequence operations
    - Indexing and slicing of strings
        - ✓ Accessing subsets of strings

```python
greetings = "Hello World"

print(greetings[0:5:1])      # Print the first five characters

print(greetings[6:11:1])     # Print the last five characters

print(greetings[0:11:2])     # Print the 1st, 3rd, ... characters
```

```
Hello
World
HloWrd
```

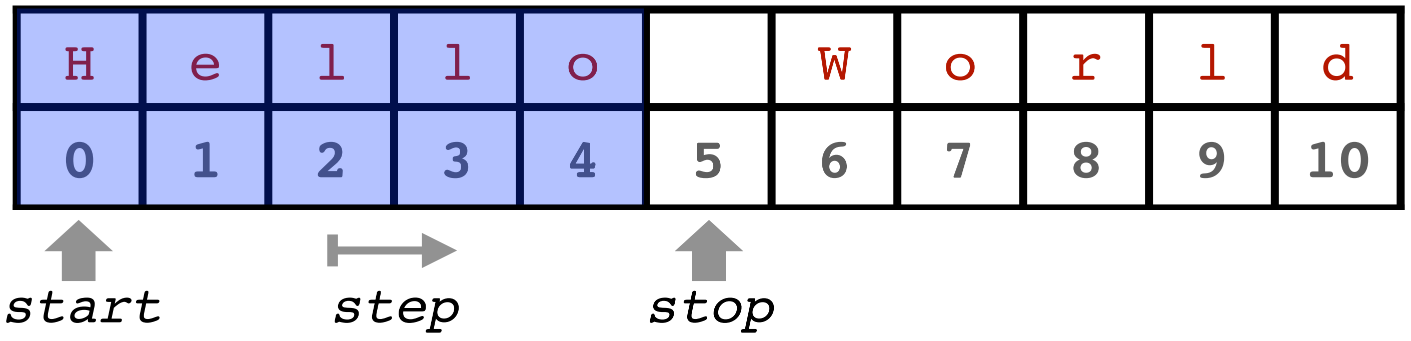
H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing subsets of strings

```
greetings = "Hello World"

print(greetings[0:5:1])      # Print the first five characters

print(greetings[6:11:1])     # Print the last five characters

print(greetings[0:11:2])     # Print the 1st, 3rd, ... characters
```

```
Hello
World
HloWrd
```

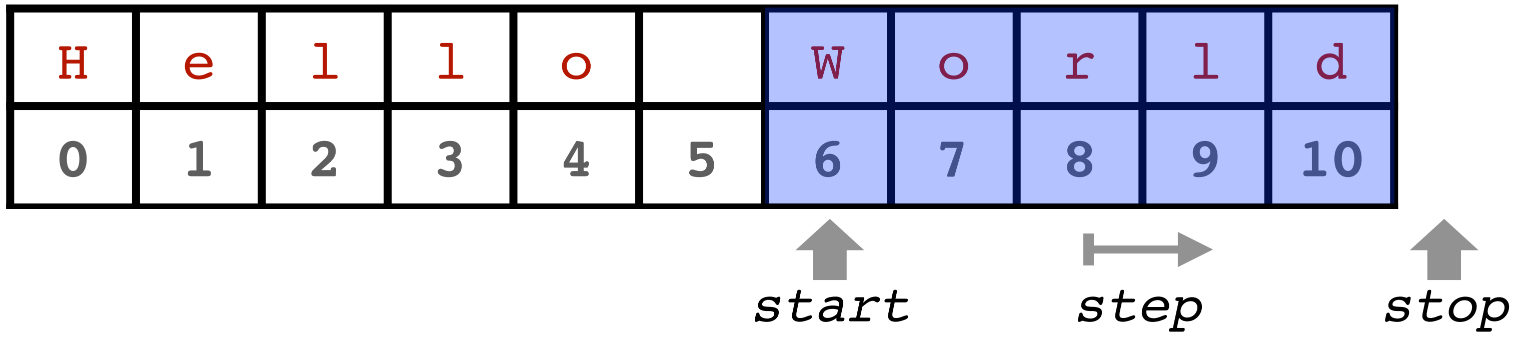


Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing subsets of strings

```
greetings = "Hello World"

print(greetings[0:5:1])      # Print the first five characters

print(greetings[6:11:1])     # Print the last five characters

print(greetings[0:11:2])     # Print the 1st, 3rd, ... characters
```

```
Hello
World
HloWrd
```

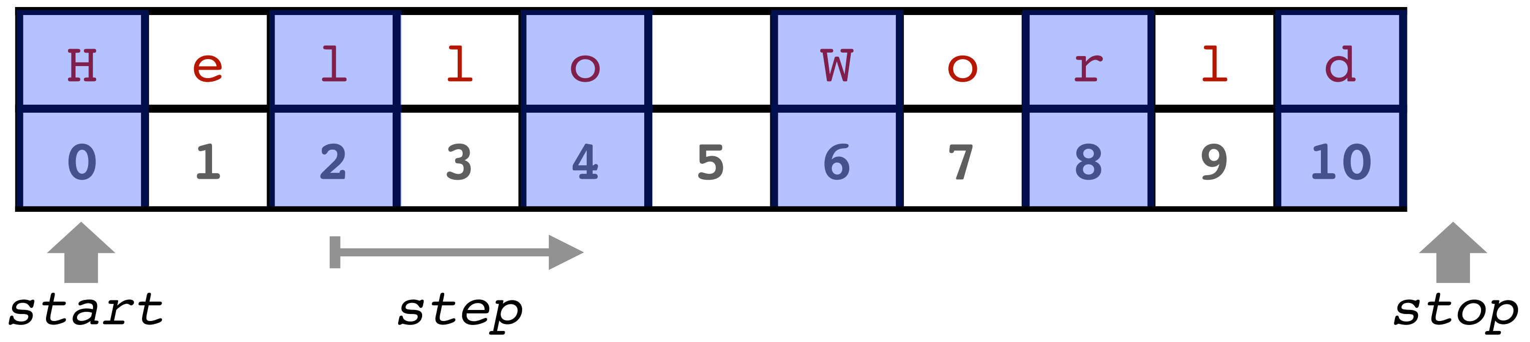


Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing subsets of strings

```
greetings = "Hello World"

print(greetings[0:5:1])      # Print the first five characters

print(greetings[6:11:1])     # Print the last five characters

print(greetings[0:11:2])     # Print the 1st, 3rd, ... characters
```

```
Hello
World
HloWrd
```



Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing subsets of strings

Arguments	Default Values
`start`	0
`stop`	length of the string
`step`	1

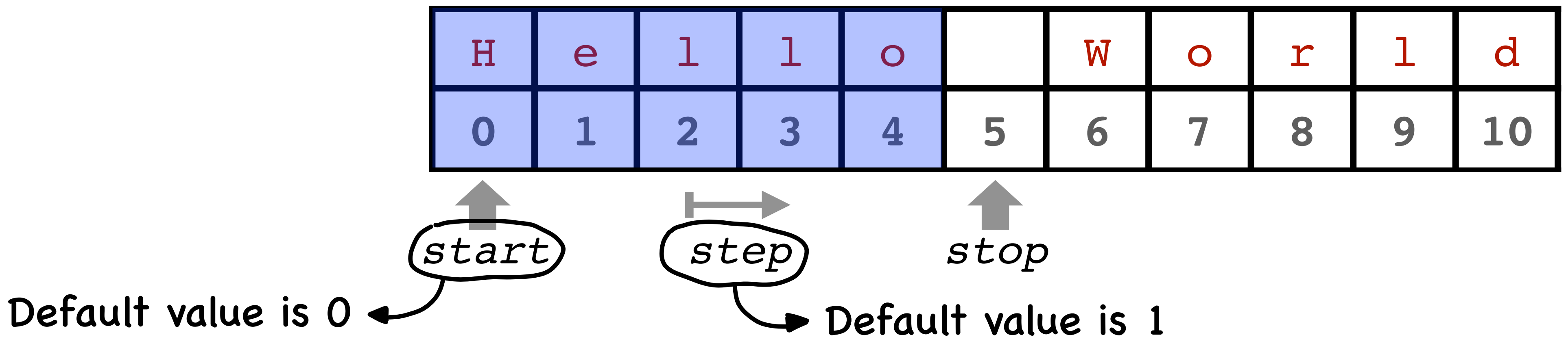
H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing subsets of strings

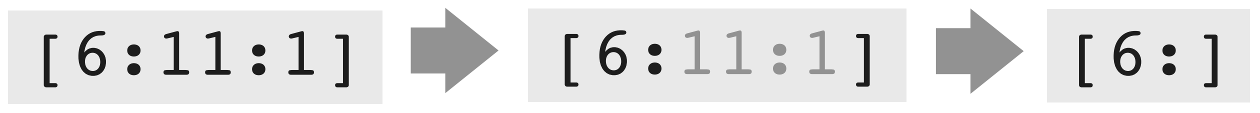
`[0:5:1]` → `[0:5:1]` → `[:5]`

Arguments	Default Values
`start`	0
`stop`	length of the string
`step`	1

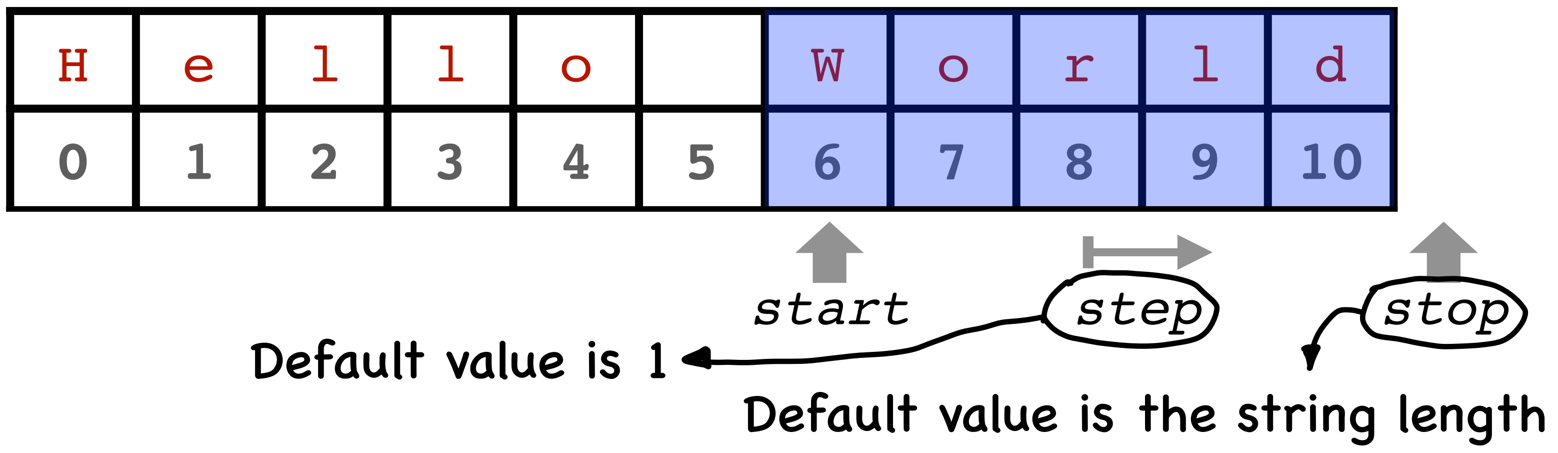


Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing subsets of strings

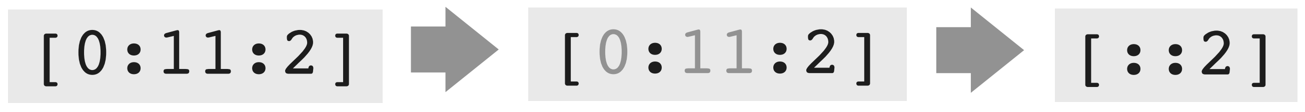


Arguments	Default Values
start	0
stop	length of the string
step	1

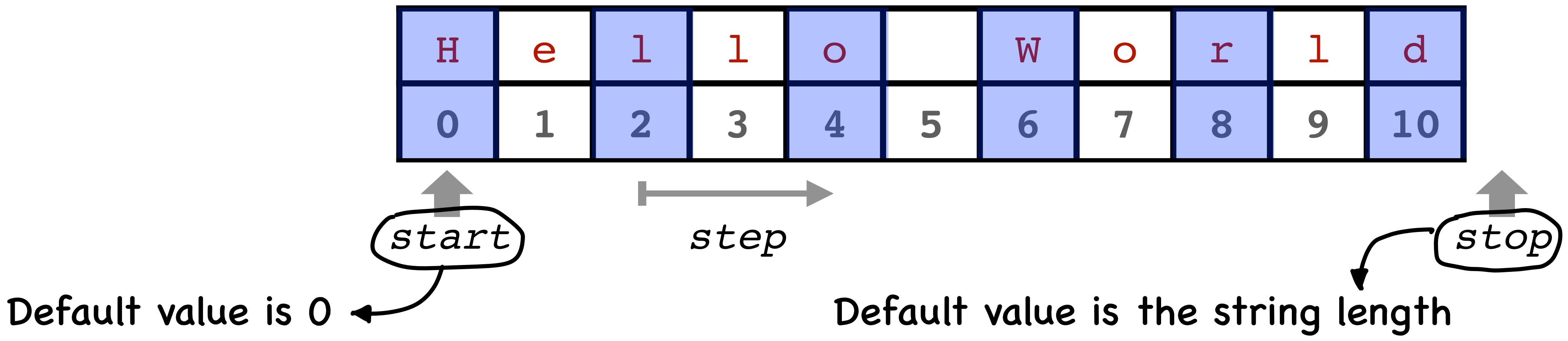


Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing subsets of strings



Arguments	Default Values
start	0
stop	length of the string
step	1

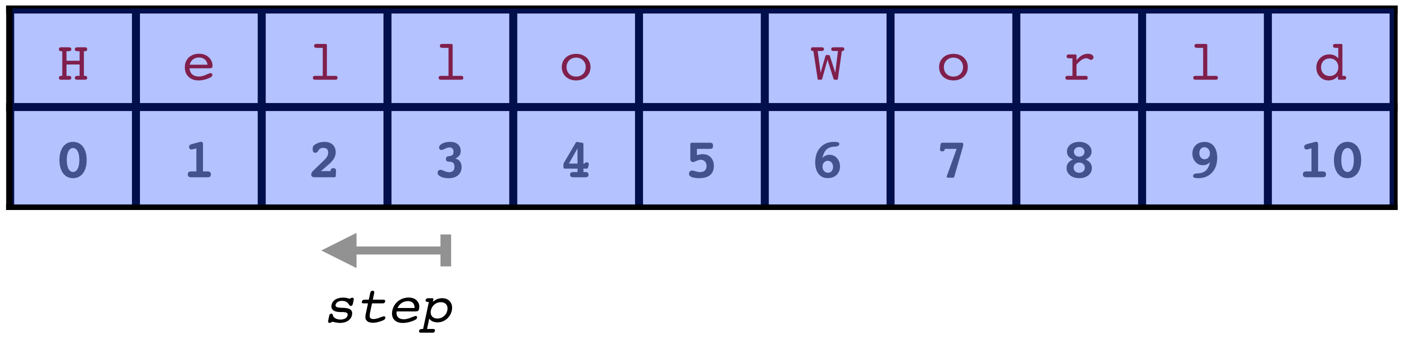


Strings

- Sequence operations
  - Indexing and slicing of strings
    - ✓ Accessing subsets of strings

```
greetings = "Hello World"

print(greetings[::-1])
```

```
dlroW olleH
```



Strings

- Methods of strings
    - A method is a special function associated with an object
    - A method is called via the syntax `object.method()`

Strings

- Methods of strings
    - Case conversion methods

```
line = "all work and no play makes Jack a dull boy"

line_upper = line.upper()
line_lower = line.lower()
line_cap = line.capitalize()
line_swap = line.swapcase()
line_title = line.title()
```

Strings

- Methods of strings
  - Case conversion methods

```
line = "all work and no play makes Jack a dull boy"

line_upper = line.upper()
line_lower = line.lower()
line_cap = line.capitalize()
line_swap = line.swapcase()
line_title = line.title()
```

```
ALL WORK AND NO PLAY MAKES JACK A DULL BOY
```

Strings

- Methods of strings
  - Case conversion methods

```python
line = "all work and no play makes Jack a dull boy"

line_upper = line.upper()
line_lower = line.lower()
line_cap = line.capitalize()
line_swap = line.swapcase()
line_title = line.title()
```

```
all work and no play makes jack a dull boy
```

Strings

- Methods of strings
  - Case conversion methods

```
line = "all work and no play makes Jack a dull boy"

line_upper = line.upper()
line_lower = line.lower()
line_cap = line.capitalize()
line_swap = line.swapcase()
line_title = line.title()
```

```
All work and no play makes jack a dull boy
```

Strings

- Methods of strings
  - Case conversion methods

```
line = "all work and no play makes Jack a dull boy"

line_upper = line.upper()
line_lower = line.lower()
line_cap = line.capitalize()
line_swap = line.swapcase()
line_title = line.title()
```

```
ALL WORK AND NO PLAY MAKES jACK A DULL BOY
```

Strings

- Methods of strings
    - Case conversion methods

```
line = "all work and no play makes Jack a dull boy"

line_upper = line.upper()
line_lower = line.lower()
line_cap = line.capitalize()
line_swap = line.swapcase()
line_title = line.title()
```

```
All Work And No Play Makes Jack A Dull Boy
```

Strings

- Methods of strings
    - Case conversion methods

Example 1: Write a program to count the number of letter "a"s (either upper case or lower case) in a given string.

```
string = """
Many years later, as he faced the firing squad, Colonel
Aureliano Buendía was to remember that distant afternoon
when his father took him to discover ice. At that time
Macondo was a village of twenty adobe houses, built on
the bank of a river of clear water that ran along a bed
of polished stones, which were white and enormous, like
prehistoric eggs.
"""
```

Strings

- Methods of strings
  - Case conversion methods

Example 1: Write a program to count the number of letter “a”s (either upper case or lower case) in a given string.

```
count = 0
for char in string:
    if char.lower() == 'a':
        count += 1

print(count)
```

Convert all letters to lower case

```
30
```

Strings

- Methods of strings
  - The `count()` method

```
string = """
Many years later, as he faced the firing squad, Colonel
Aureliano Buendía was to remember that distant afternoon
when his father took him to discover ice. At that time
Macondo was a village of twenty adobe houses, built on
the bank of a river of clear water that ran along a bed
of polished stones, which were white and enormous, like
prehistoric eggs.
"""
```

```
count = string.lower().count('a')
print(count)
```

```
30
```

Strings

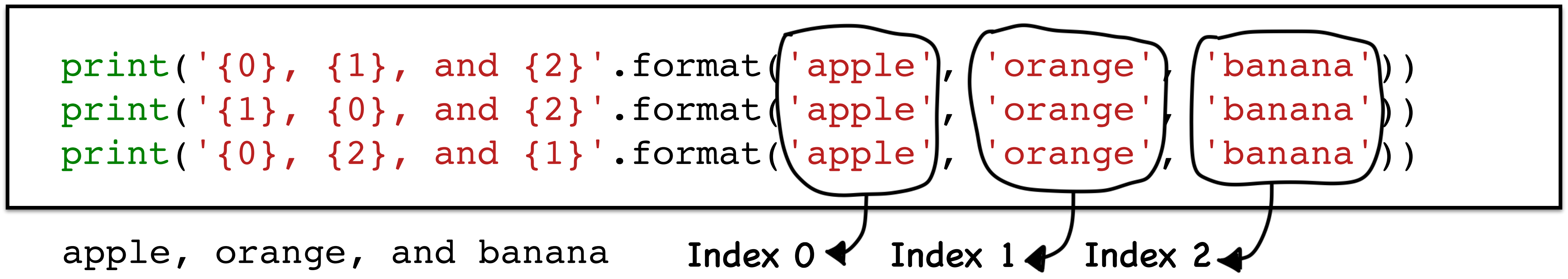
- Methods of strings
  - The `format()` method

```python
exam = 85                            # Final exam marks of a course
grade = 'A+'                         # Final grade of the course

text = 'Your exam marks: {}, your grade: {}'.format(exam, grade)
print(text)
```

```
Your exam marks: 85, your grade: A+
```

Strings

- Methods of strings
  - The `format()` method



```
print('{0}, {1}, and {2}'.format('apple', 'orange', 'banana'))
print('{1}, {0}, and {2}'.format('apple', 'orange', 'banana'))
print('{0}, {2}, and {1}'.format('apple', 'orange', 'banana'))
```

```
apple, orange, and banana
orange, apple, and banana
apple, banana, and orange
```

Strings

- Methods of strings
  - The `format()` method

```
name = 'John'
balance = 25678.95

print(f'Hello {name}, you have ${balance} in your account.')
```

Values can be inserted to the curly brackets in the f-string

```
Hello John, you have $25678.95 in your account.
```

Strings

- Methods of strings
  - The `replace()` method

```
string_us = 'Coffee enhances my modeling skills.'
print(string_us)

string_uk = string_us.replace('modeling', 'modelling')
print(string_uk)

string_sg = string_uk.replace('Coffee', 'Kopi')
print(string_sg)
```

```
Coffee enhances my modeling skills.
Coffee enhances my modelling skills.
Kopi enhances my modelling skills.
```

Strings

- Methods of strings
  - The `replace()` method

```python
string_us = 'Coffee enhances my modeling skills.'
print(string_us)

string_uk = string_us.replace('modeling', 'modelling')
print(string_uk)

string_sg = string_uk.replace('Coffee', 'Kopi')
print(string_sg)
```

```
Coffee enhances my modeling skills.
Coffee enhances my modelling skills.
Kopi enhances my modelling skills.
```

Strings

- Methods of strings
  - The `replace()` method

```python
string_us = 'Coffee enhances my modeling skills.'
print(string_us)

string_uk = string_us.replace('modeling', 'modelling')
print(string_uk)

string_sg = string_uk.replace('Coffee', 'Kopi')
print(string_sg)
```

```
Coffee enhances my modeling skills.
Coffee enhances my modelling skills.
Kopi enhances my modelling skills.
```

Strings

- Methods of strings
  - The `replace()` method

```
string_us = 'Coffee enhances my modeling skills.'
print(string_us)

string_uk = string_us.replace('modeling', 'modelling')
print(string_uk)

string_sg = string_uk.replace('Coffee', 'Kopi')
print(string_sg)
```

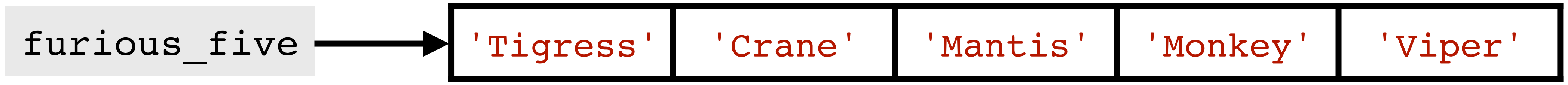
```
Coffee enhances my modeling skills.
Coffee enhances my modelling skills.
Kopi enhances my modelling skills.
```

Lists

- Create lists
  - Data items enclosed in square brackets and separated by commas

```python
furious_five = ['Tigress', 'Crane', 'Mantis', 'Monkey', 'Viper']
print(furious_five)
print(type(furious_five))
```

```
['Tigress', 'Crane', 'Mantis', 'Monkey', 'Viper']
<class 'list'>
```

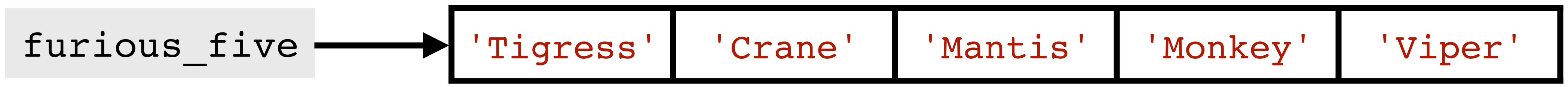


Lists

- Create lists
  - Data items enclosed in square brackets and separated by commas

```
furious_five = ['Tigress', 'Crane', 'Mantis', 'Monkey', 'Viper']
print(furious_five)
print(type(furious_five))
```

```
['Tigress', 'Crane', 'Mantis', 'Monkey', 'Viper']
<class 'list'>
```



Lists

- Create lists
  - Data items enclosed in square brackets and separated by commas

```python
numbers = [1, 2.0, 3.0, 4, 5, 6.0]
print(numbers)
```

```
[1, 2.0, 3.0, 4, 5, 6.0]
```

Lists

- Create lists
  - Data items enclosed in square brackets and separated by commas

```python
condo = ['SEASCAPE',       # Name of the condo project (str)
         'CCR',            # Region segment of the condo (str)
         'Resale',         # Sale type of the condo (str)
         4388000.0]        # Price of the condo (float)
print(condo)
```

```
['SEASCAPE', 'CCR', 'Resale', 4388000.0]
```

Coding Style: Limit all lines of code to a maximum of 79 characters.

The preferred way of wrapping long lines is by using Python's implied line continuation inside parentheses, brackets and braces. Long lines can be broken over multiple lines by wrapping expressions in parentheses. —**PEP 8 Style Guide**

Lists

- Create lists
  - Other cases
    - ✓ Empty lists

```
feel_empty = []
print(feel_empty)
```

```
[]
```

    - ✓ Data type conversions

```
print(list('abcd'))      # Convert a string into a list
print(list(range(5)))    # Convert the range type object into a list
```

```
['a', 'b', 'c', 'd']
[0, 1, 2, 3, 4]
```

Lists

- Comparison between lists and strings
  - Similarities
    - ✓ The same `len()` function
    - ✓ The same indexing and slicing system

`"Hello"`

5 characters

H	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

`['Tigress', 'Crane', 'Mantis', 'Monkey', 'Viper']`

5 data items

'Tigress'	'Crane'	'Mantis'	'Monkey'	'Viper'
0	1	2	3	4
-5	-4	-3	-2	-1

Lists

- Comparison between lists and strings
  - Similarities
    - ✓ The same `len()` function
    - ✓ The same indexing and slicing system

```python
last_warrior = furious_five[-1]
first_two_warriors = furious_five[:2]
```

```
Viper
['Tigress', 'Crane']
```

'Tigress'	'Crane'	'Mantis'	'Monkey'	'Viper'
0	1	2	3	4
-5	-4	-3	-2	-1

Lists

- Comparison between lists and strings
  - Similarities
    - ✓ The same `len()` function
    - ✓ The same indexing and slicing system

```
last_warrior = furious_five[-1]
first_two_warriors = furious_five[:2]
```

```
Viper
['Tigress', 'Crane']
```

'Tigress'	'Crane'	'Mantis'	'Monkey'	'Viper'
0	1	2	3	4
-5	-4	-3	-2	-1

Lists

- Comparison between lists and strings
  - Similarities

✓ The same `len()` function

✓ The same indexing and slicing system

```
last_warrior = furious_five[-1]
first_two_warriors = furious_five[:2]
```

```
Viper
['Tigress', 'Crane']
```

'Tigress'	'Crane'	'Mantis'	'Monkey'	'Viper'
0	1	2	3	4
-5	-4	-3	-2	-1

Lists

- Comparison between lists and strings
  - Similarities

    ✓ The same operators `+` and `*`

```
letters = ['A', 'B', 'C']
numbers = [2, 2.5]

mixed = letters + numbers*3

print(mixed)
print(len(mixed))
```

```
['A', 'B', 'C', 2, 2.5, 2, 2.5, 2, 2.5]
9
```

Lists

- Comparison between lists and strings
  - Similarities
    - ✓ The same operators `+` and `*`

```
letters = ['A', 'B', 'C']
numbers = [2, 2.5]

mixed = letters + numbers*3

print(mixed)
print(len(mixed))
```

```
['A', 'B', 'C', 2, 2.5, 2, 2.5, 2, 2.5]
9
```

Lists

- Comparison between lists and strings
  - Similarities

    ✓ The same operators `+` and `*`

```
letters = ['A', 'B', 'C']
numbers = [2, 2.5]

mixed = letters + numbers*3

print(mixed)
print(len(mixed))
```

```
['A', 'B', 'C', 2, 2.5, 2, 2.5, 2, 2.5]
9
```

Lists

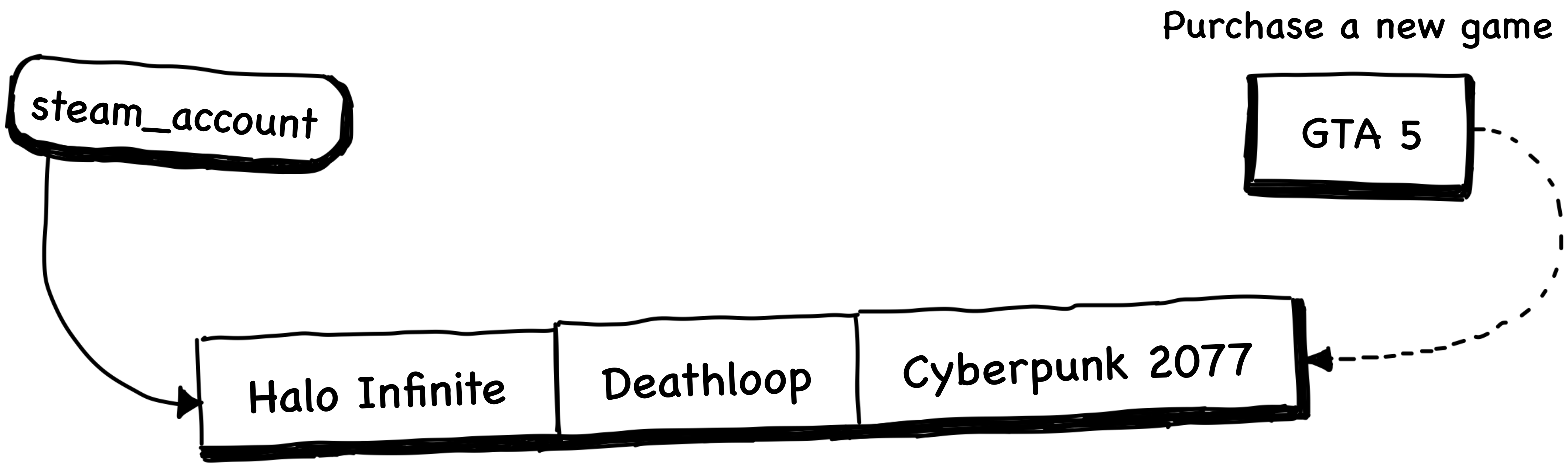
- Comparison between lists and strings
  - Similarities
    - ✓ The same operators `+` and `*`

```
letters = ['A', 'B', 'C']
numbers = [2, 2.5]

mixed = letters + numbers*3

print(mixed)
print(len(mixed))
```

```
['A', 'B', 'C', 2, 2.5, 2, 2.5, 2, 2.5]
9
```

Lists

- Comparison between lists and strings
  - Differences
    - ✓ Mutability: can be changed in-place



Lists

- Comparison between lists and strings
  - Differences
    - ✓ Mutability: can be changed in-place

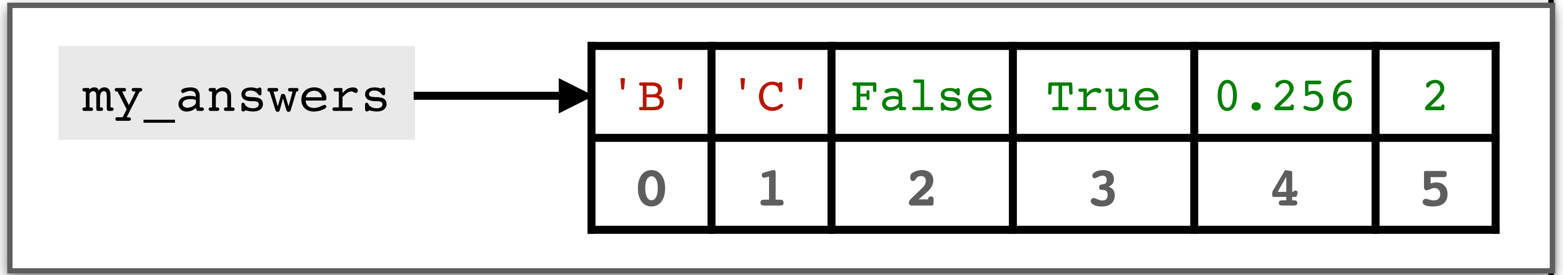
```
my_answers = ['B', 'C', False, True, 0.256, 2]
print(my_answers)

my_answers[1] = 'D'
print(my_answers)

my_answers[2:4] = [True, False]
print(my_answers)
```

```
['B', 'C', False, True, 0.256, 2]
['B', 'D', False, True, 0.256, 2]
['B', 'D', True, False, 0.256, 2]
```

Lists

- Comparison between lists and strings
  - Differences
    - ✓ Mutability: can be changed in-place

```
my_answers = ['B', 'C', False, True, 0.256, 2]
print(my_answers)

my_answers[1] = 'D'
print(my_answers)

my_answers[2:4] = [True, False]
print(my_answers)
```

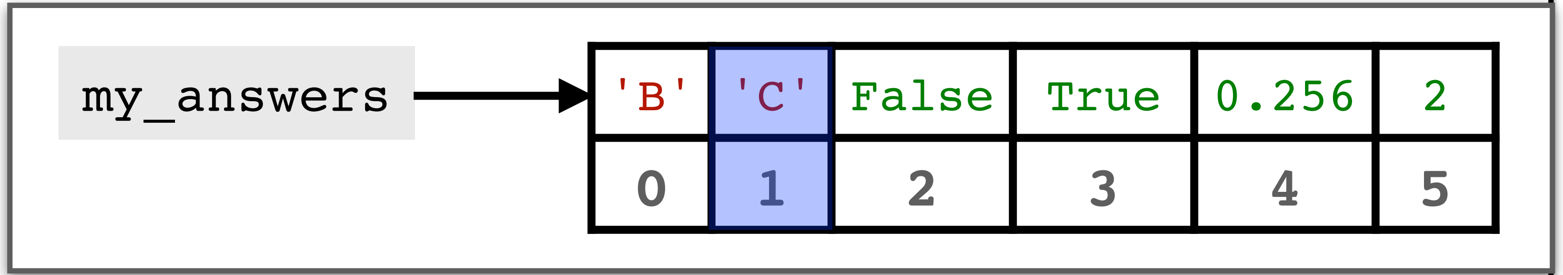


```
['B', 'C', False, True, 0.256, 2]
['B', 'D', False, True, 0.256, 2]
['B', 'D', True, False, 0.256, 2]
```

Lists

- Comparison between lists and strings
  - Differences
    - ✓ Mutability: can be changed in-place

```
my_answers = ['B', 'C', False, True, 0.256, 2]
print(my_answers)

my_answers[1] = 'D'
print(my_answers)

my_answers[2:4] = [True, False]
print(my_answers)
```

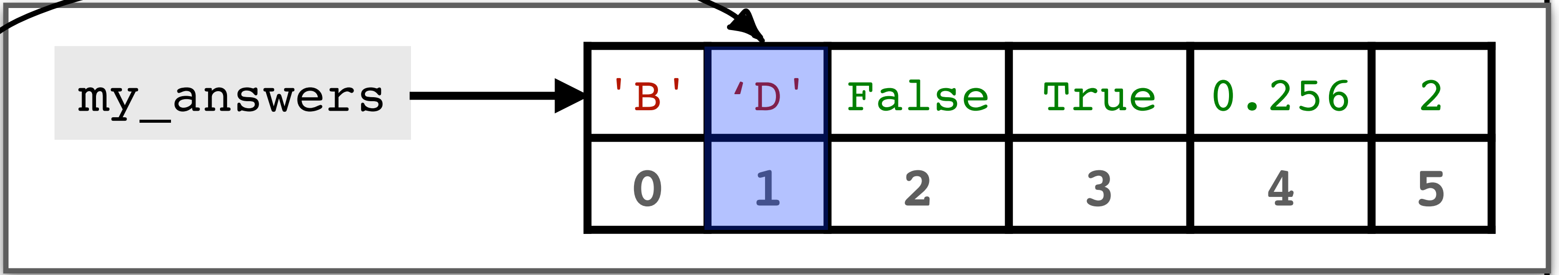


```
['B', 'C', False, True, 0.256, 2]
['B', 'D', False, True, 0.256, 2]
['B', 'D', True, False, 0.256, 2]
```

Lists

- Comparison between lists and strings
  - Differences
    - ✓ Mutability: can be changed in-place

```
my_answers = ['B', 'C', False, True, 0.256, 2]
print(my_answers)

my_answers[1] = 'D'
print(my_answers)

my_answers[2:4] = [True, False]
print(my_answers)
```

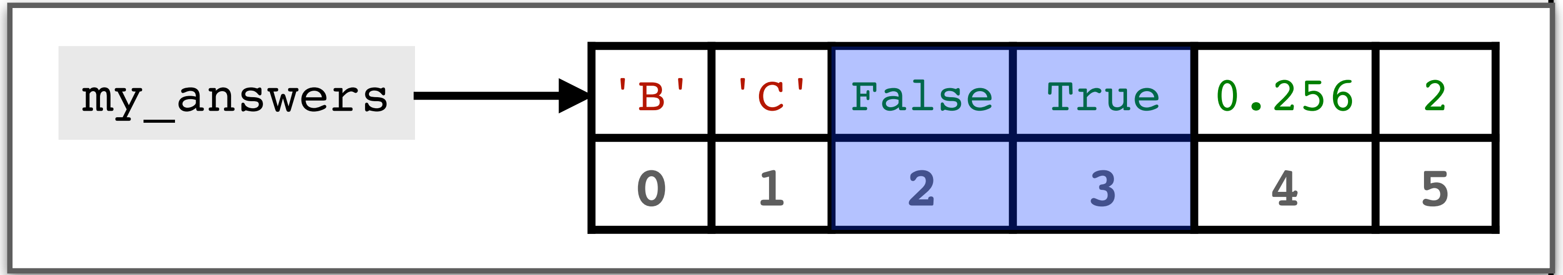


```
['B', 'C', False, True, 0.256, 2]
['B', 'D', False, True, 0.256, 2]
['B', 'D', True, False, 0.256, 2]
```

Lists

- Comparison between lists and strings
  - Differences
    - ✓ Mutability: can be changed in-place

```
my_answers = ['B', 'C', False, True, 0.256, 2]
print(my_answers)

my_answers[1] = 'D'
print(my_answers)

my_answers[2:4] = [True, False]
print(my_answers)
```

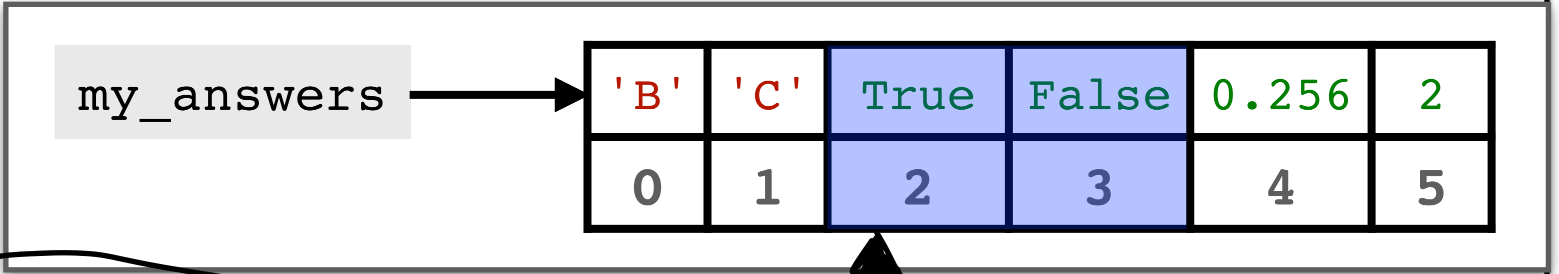


```
['B', 'C', False, True, 0.256, 2]
['B', 'D', False, True, 0.256, 2]
['B', 'D', True, False, 0.256, 2]
```

Lists

- Comparison between lists and strings
  - Differences
    - ✓ Mutability: can be changed in-place

```
my_answers = ['B', 'C', False, True, 0.256, 2]
print(my_answers)

my_answers[1] = 'D'
print(my_answers)

my_answers[2:4] = [True, False]
print(my_answers)
```



```
['B', 'C', False, True, 0.256, 2]
['B', 'D', False, True, 0.256, 2]
['B', 'D', True, False, 0.256, 2]
```

Lists

- List methods
  - Adding items by `append()`, `extend()`, and `insert()`

```
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077']
print(games)

games.append('AOE IV')
print(games)

other_games = ['Battlefield 2042’, 'Fifa 22']
games.extend(other_games)
print(games)
```

```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077']
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV',
'Battlefield 2042', 'Fifa 22']
```

Lists

- List methods
  - Adding items by `append()`, `extend()`, and `insert()`

```
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077']
print(games)

games.append('AOE IV')
print(games)

other_games = ['Battlefield 2042’, 'Fifa 22']
games.extend(other_games)
print(games)
```

```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077']
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV',
'Battlefield 2042', 'Fifa 22']
```

Lists

- List methods
  - Adding items by `append()`, `extend()`, and `insert()`

```python
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077']
print(games)

games.append('AOE IV')
print(games)

other_games = ['Battlefield 2042’, 'Fifa 22']
games.extend(other_games)
print(games)
```

Appended to the end of the list

```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077']
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV',
'Battlefield 2042', 'Fifa 22']
```

Lists

- List methods
  - Adding items by `append()`, `extend()`, and `insert()`

```
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077']
print(games)

games.append('AOE IV')
print(games)

other_games = ['Battlefield 2042', 'Fifa 22']
games.extend(other_games)
print(games)
```

Multiple items appended to the end of the list

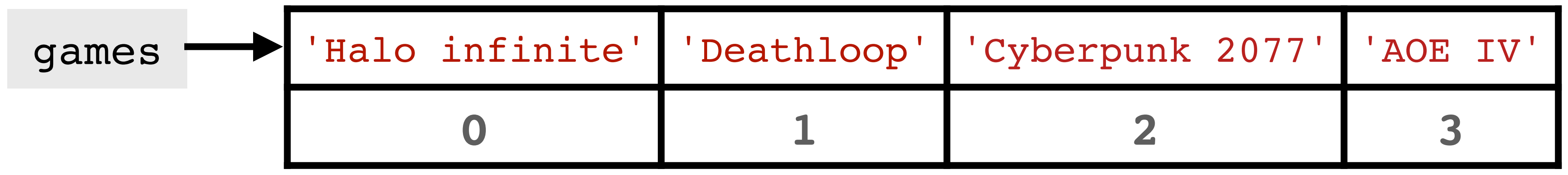
```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077']
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV',
'Battlefield 2042', 'Fifa 22']
```

Lists

- List methods
  - Adding items by `append()`, `extend()`, and `insert()`

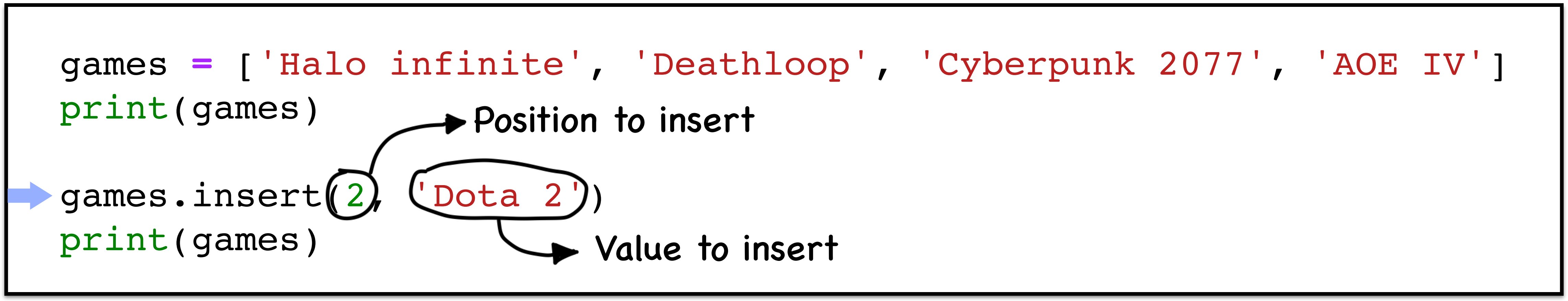
```python
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
print(games)

games.insert(2, 'Dota 2')
print(games)
```

```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
['Halo infinite', 'Deathloop', 'Dota 2', 'Cyberpunk 2077',
'AOE IV']
```



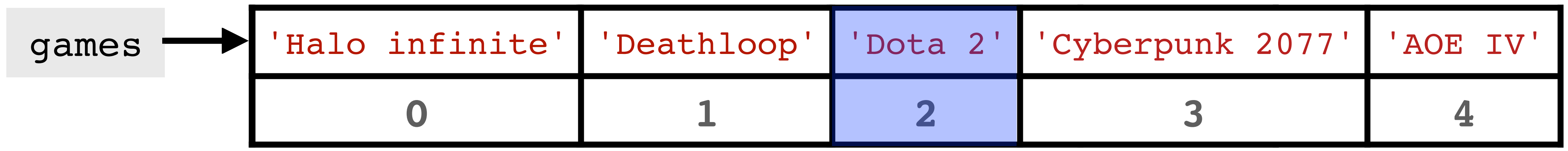
Lists

- List methods
  - Adding items by `append()`, `extend()`, and `insert()`



```
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
print(games)

games.insert(2, 'Dota 2')
print(games)
```

```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
['Halo infinite', 'Deathloop', 'Dota 2', 'Cyberpunk 2077',
'AOE IV']
```



`games` →

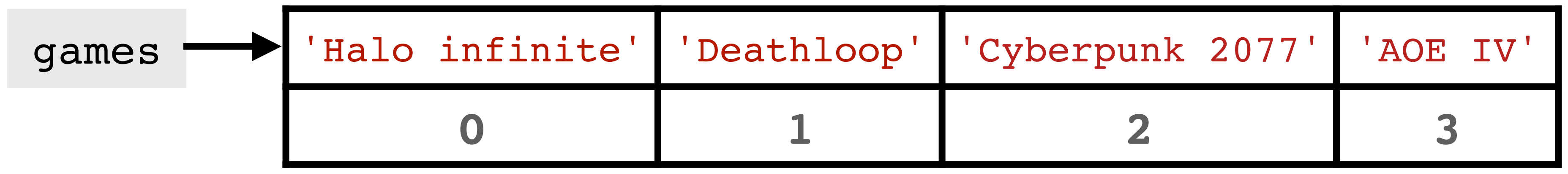
'Halo infinite'	'Deathloop'	'Dota 2'	'Cyberpunk 2077'	'AOE IV'
0	1	2	3	4

Lists

- List methods
  - Deleting items by the `remove()` method

```
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
print(games)

games.remove('Deathloop')
print(games)
```

```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
['Halo infinite', 'Cyberpunk 2077', 'AOE IV']
```

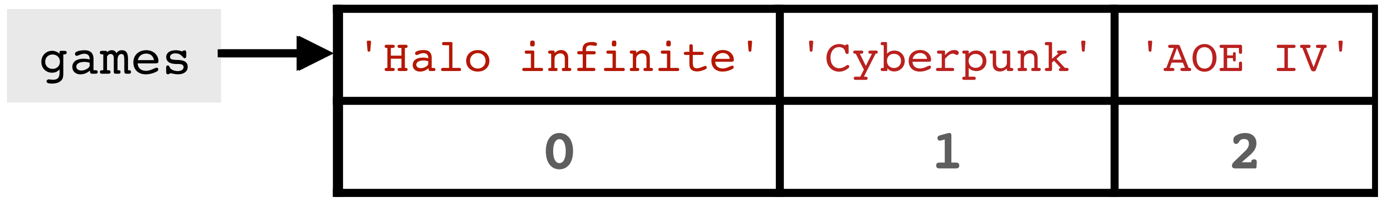


Lists

- List methods
  - Deleting items by the `remove()` method

```python
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
print(games)

games.remove('Deathloop')
print(games)
```

```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
['Halo infinite', 'Cyberpunk 2077', 'AOE IV']
```



Lists

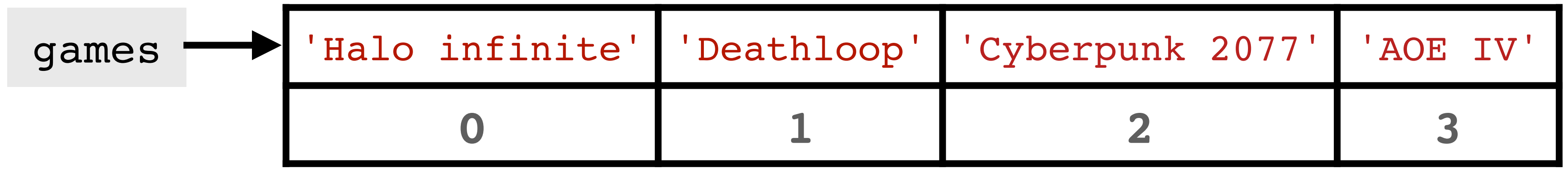
- List methods
  - Deleting items by the `remove()` method
    - ✓ Remove the first appearance
    - ✓ An error message is raised if the given value does not appear in the list

Lists

- List methods
  - Deleting items by the `pop()` method

```
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
print(games)

item = games.pop(2)
print(item)
print(games)
```

```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
Cyberpunk 2077
['Halo infinite', 'Deathloop', 'AOE IV']
```

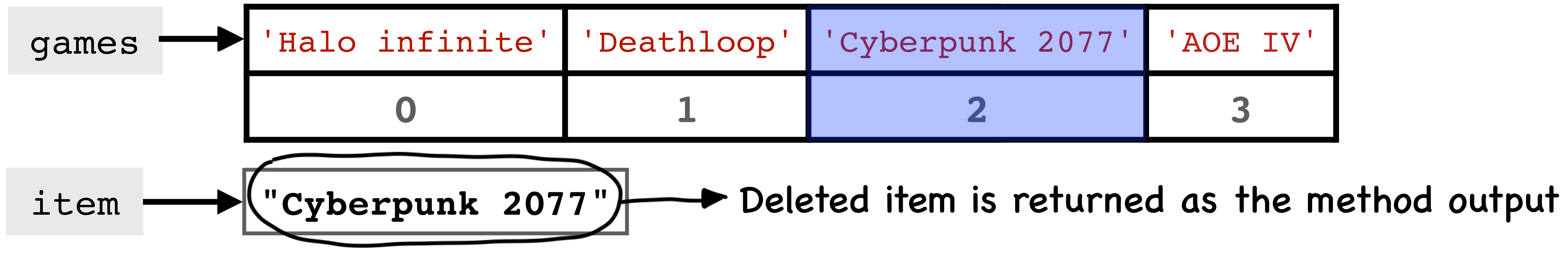


Lists

- List methods
  - Deleting items by the `pop()` method

```python
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
print(games)

item = games.pop(2)
print(item)
print(games)
```

```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
Cyberpunk 2077
['Halo infinite', 'Deathloop', 'AOE IV']
```

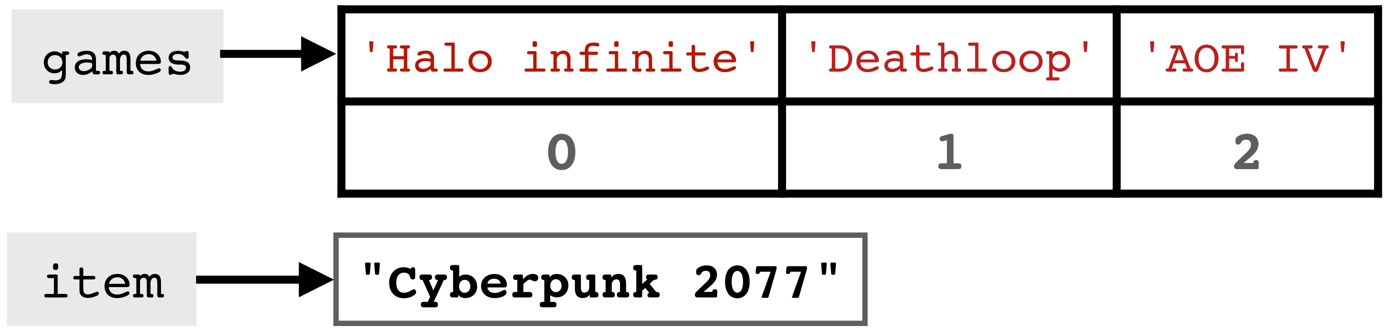


Lists

- List methods
  - Deleting items by the `pop()` method

```python
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
print(games)

item = games.pop(2)
print(item)
print(games)
```

```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
Cyberpunk 2077
['Halo infinite', 'Deathloop', 'AOE IV']
```

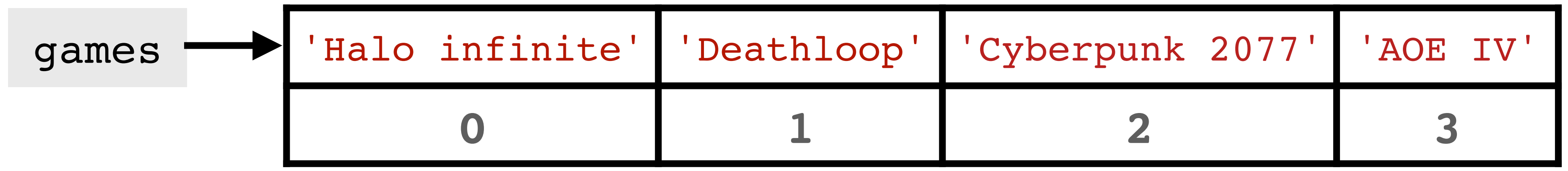


Lists

- List methods
  - Deleting items by the `pop()` method

```python
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
print(games)

item = games.pop()
print(item)
print(games)
```

```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
AOE IV
['Halo infinite', 'Deathloop', ‘Cyberpunk 2077’]
```



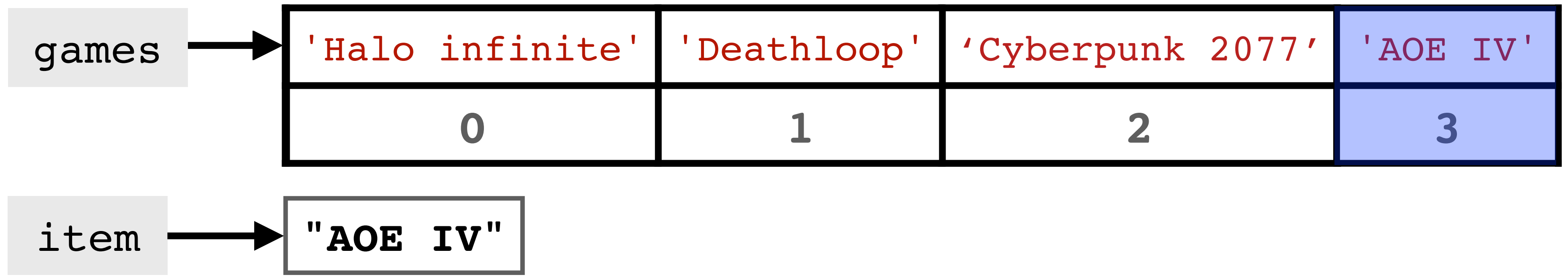
Lists

- List methods
  - Deleting items by the `pop()` method

```
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
print(games)

item = games.pop()
print(item)
print(games)
```

Remove and return the last item if index is not specified

```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
AOE IV
['Halo infinite', 'Deathloop', 'Cyberpunk 2077']
```

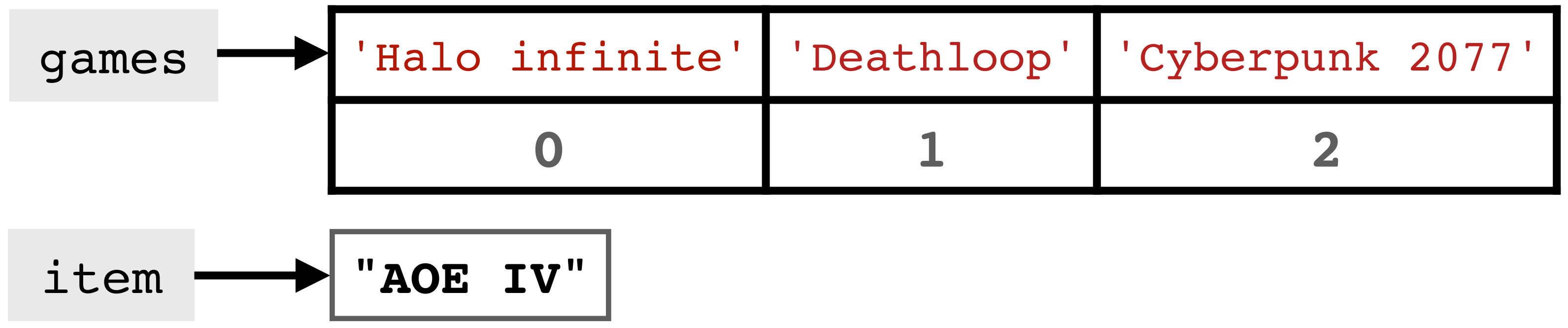


Lists

- List methods
  - Deleting items by the `pop()` method

```
games = ['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
print(games)

item = games.pop()
print(item)
print(games)
```

```
['Halo infinite', 'Deathloop', 'Cyberpunk 2077', 'AOE IV']
AOE IV
['Halo infinite', 'Deathloop', ‘Cyberpunk 2077’]
```



Lists

- List methods
  - Searching for an item by `index()`

```
num_list = [3.5, 2.6, 0.2, 3.30, 1.8, 2.9, 5, 3.3]

print(num_list.index(3.3))
```

```
3
```

Lists

- List methods
  - Searching for an item by `index()`

```
num_list = [3.5, 2.6, 0.2, 3.30, 1.8, 2.9, 5, 3.3]

print(num_list.index(3.3))
```

Other appearances are ignored

Return the index of the first appearance of the given value

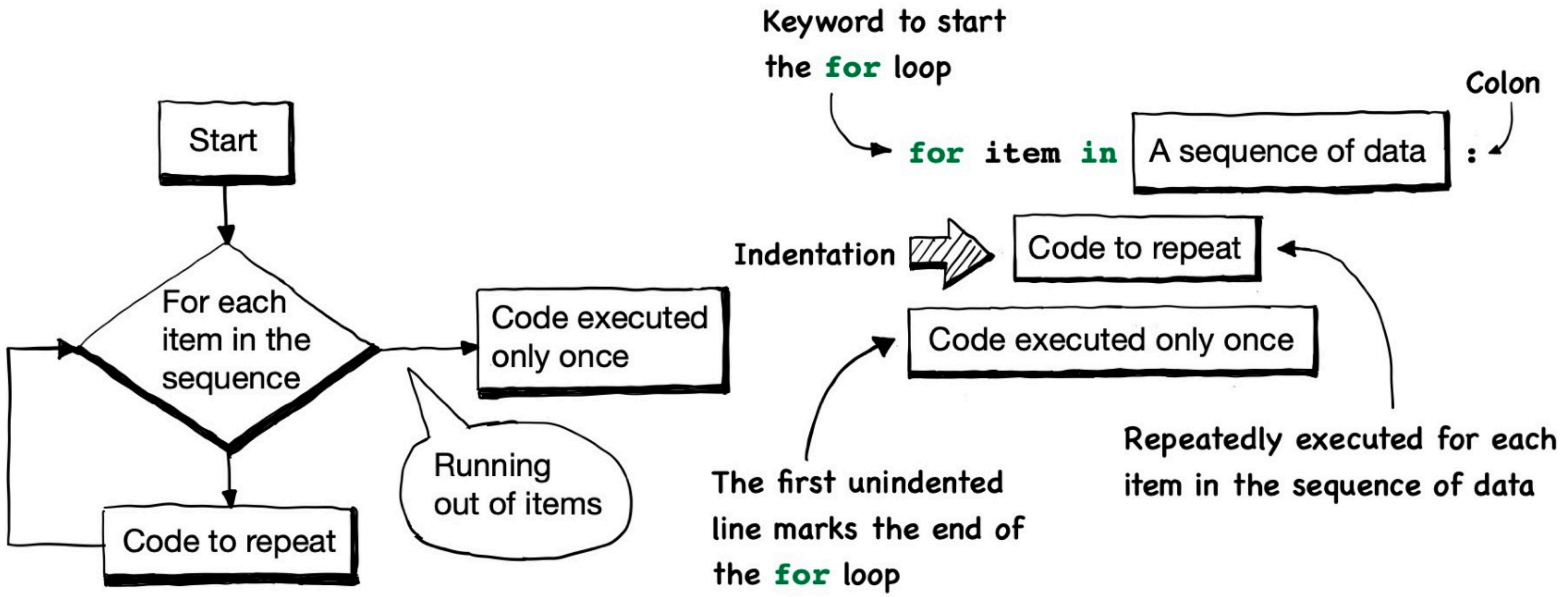
```
3
```

Lists

- List methods
  - Searching for an item by `index()`
    - ✓ Index of the first appearance of the given value
    - ✓ An error message is raised if the given value does not appear in the list

Lists

- Lists as iterables
  - Iterables
    - ✓ Each element is returned in an iteration of a `for` loop


Start
For each item in the sequence
Code executed only once
Running out of items
Code to repeat
Keyword to start the for loop
Colon
for item in
A sequence of data
:
Indentation
Code to repeat
Code executed only once
The first unindented line marks the end of the for loop
Repeatedly executed for each item in the sequence of data

Lists

- Lists as iterables
    - Iterating list items using a `for` loop

Example 2: The **`usd`** list contains four money transactions in US dollars. Create another list named **`sgd`** that transfers each transaction into Singapore dollars.

Lists

- Lists as iterables
  - Iterating list items using a `for` loop

Example 2: The **`usd`** list contains four money transactions in US dollars. Create another list named **`sgd`** that transfers each transaction into Singapore dollars.

```python
usd = [2, 3.60, 2.05, 13.50]

for trans in usd:
    print(trans)
```

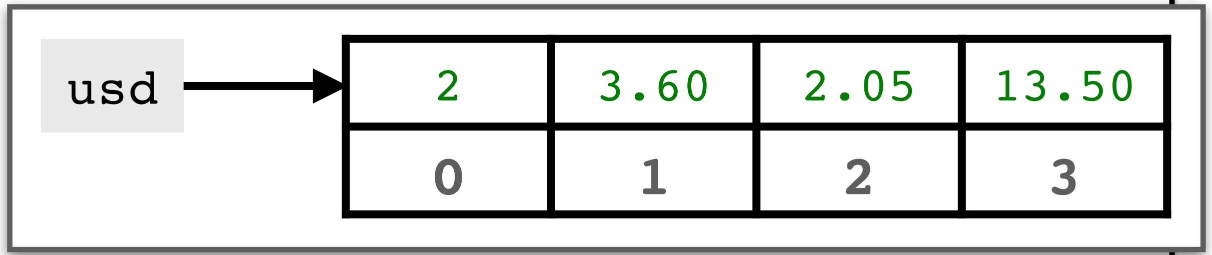
```
2
3.6
2.05
13.5
```

Lists

- Lists as iterables
  - Iterating list items using a `for` loop

Example 2: The **`usd`** list contains four money transactions in US dollars. Create another list named **`sgd`** that transfers each transaction into Singapore dollars.

```python
usd = [2, 3.60, 2.05, 13.50]

for trans in usd:
    print(trans)
```



```
2
3.6
2.05
13.5
```

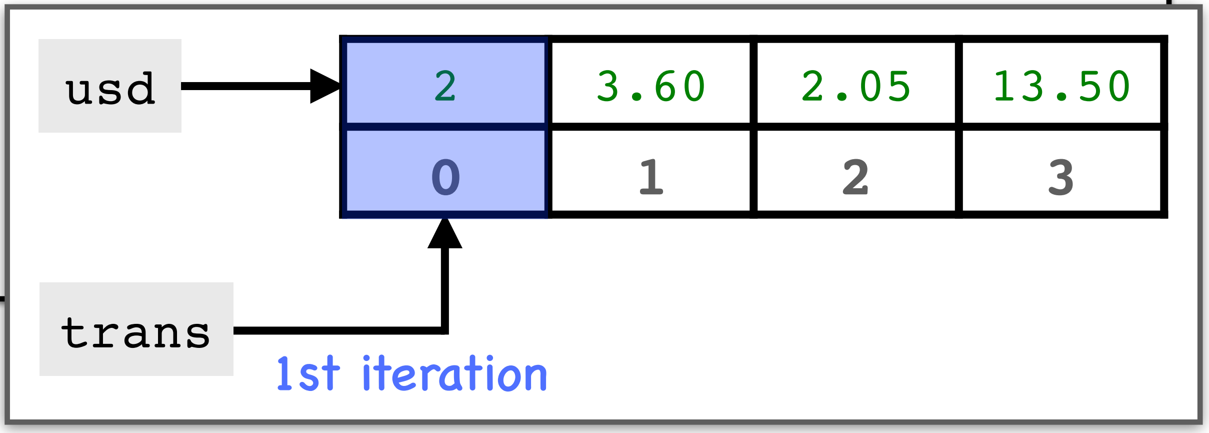
Lists

- Lists as iterables
  - Iterating list items using a `for` loop

Example 2: The **`usd`** list contains four money transactions in US dollars. Create another list named **`sgd`** that transfers each transaction into Singapore dollars.

```
usd = [2, 3.60, 2.05, 13.50]

for trans in usd:
    print(trans)
```

```
2
3.6
2.05
13.5
```



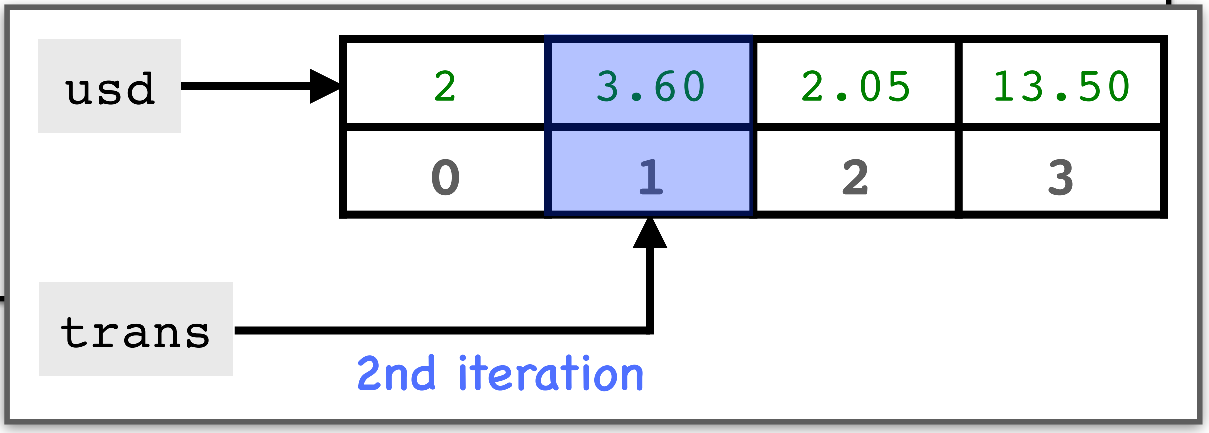
Lists

- Lists as iterables
  - Iterating list items using a `for` loop

Example 2: The **`usd`** list contains four money transactions in US dollars. Create another list named **`sgd`** that transfers each transaction into Singapore dollars.

```python
usd = [2, 3.60, 2.05, 13.50]

for trans in usd:
    print(trans)
```

```
2
3.6
2.05
13.5
```



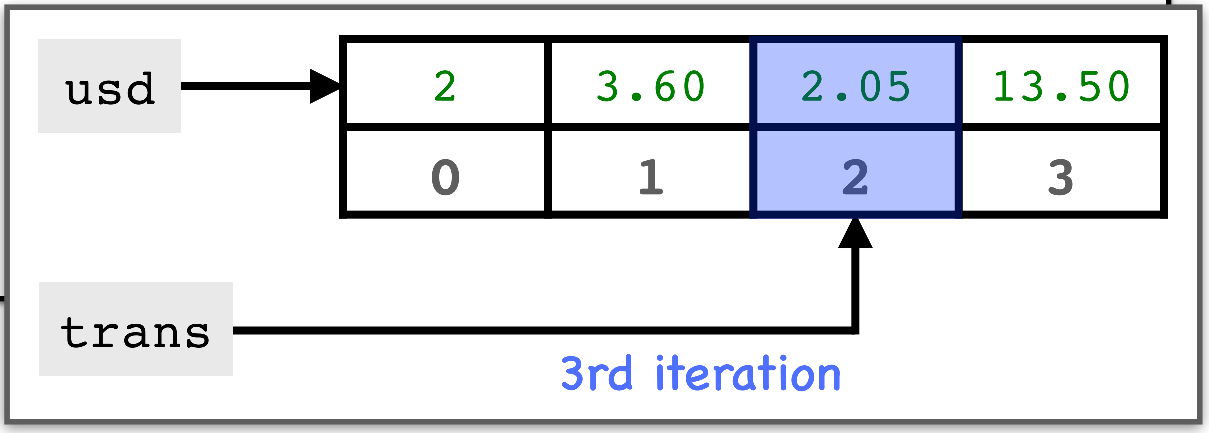
Lists

- Lists as iterables
  - Iterating list items using a `for` loop

Example 2: The **`usd`** list contains four money transactions in US dollars. Create another list named **`sgd`** that transfers each transaction into Singapore dollars.

```
usd = [2, 3.60, 2.05, 13.50]

for trans in usd:
    print(trans)
```

```
2
3.6
2.05
13.5
```



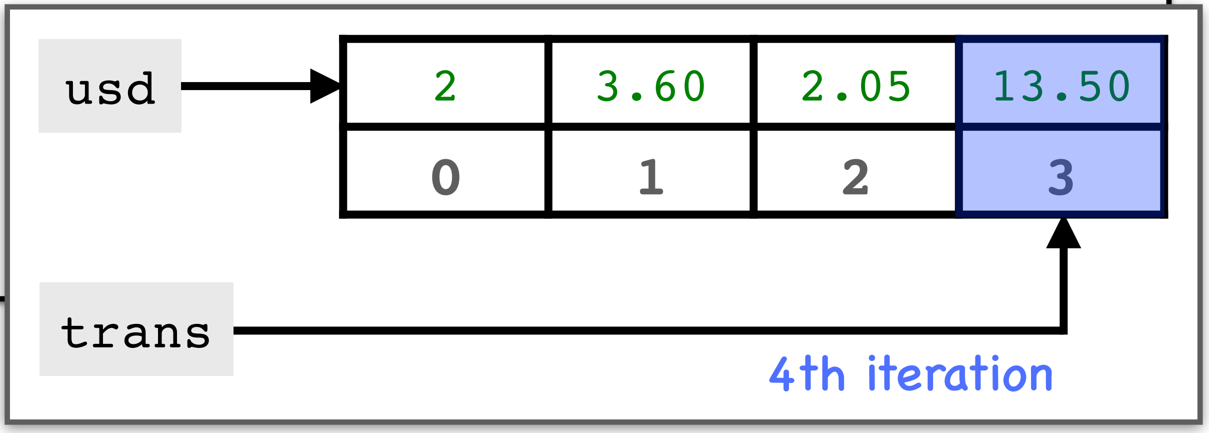
Lists

- Lists as iterables
    - Iterating list items using a `for` loop

Example 2: The **`usd`** list contains four money transactions in US dollars. Create another list named **`sgd`** that transfers each transaction into Singapore dollars.

```python
usd = [2, 3.60, 2.05, 13.50]

for trans in usd:
    print(trans)
```

```
2
3.6
2.05
13.5
```



Lists

- Lists as iterables
  - Iterating list items using a `for` loop

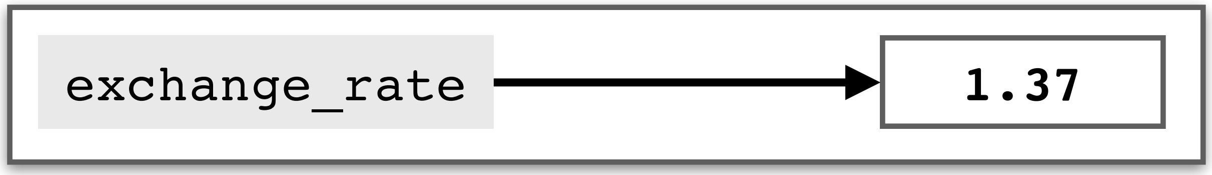
Example 2: The **`usd`** list contains four money transactions in US dollars. Create another list named **`sgd`** that transfers each transaction into Singapore dollars.

```python
exchange_rate = 1.37
usd = [2, 3.60, 2.05, 13.50]

sgd = []
for trans in usd:
    sgd.append(trans*exchange_rate)

print(sgd)
```

```
[2.74, 4.932, 2.8085, 18.495]
```

Lists

- Lists as iterables
  - Iterating list items using a `for` loop

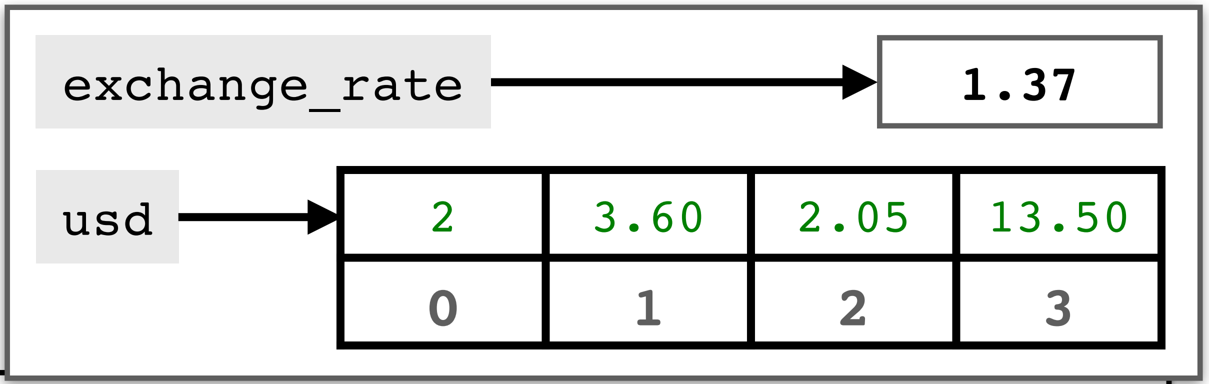


```python
exchange_rate = 1.37
usd = [2, 3.60, 2.05, 13.50]

sgd = []
for trans in usd:
    sgd.append(trans*exchange_rate)

print(sgd)
```

```
[2.74, 4.932, 2.8085, 18.495]
```

Lists

- Lists as iterables
  - Iterating list items using a `for` loop



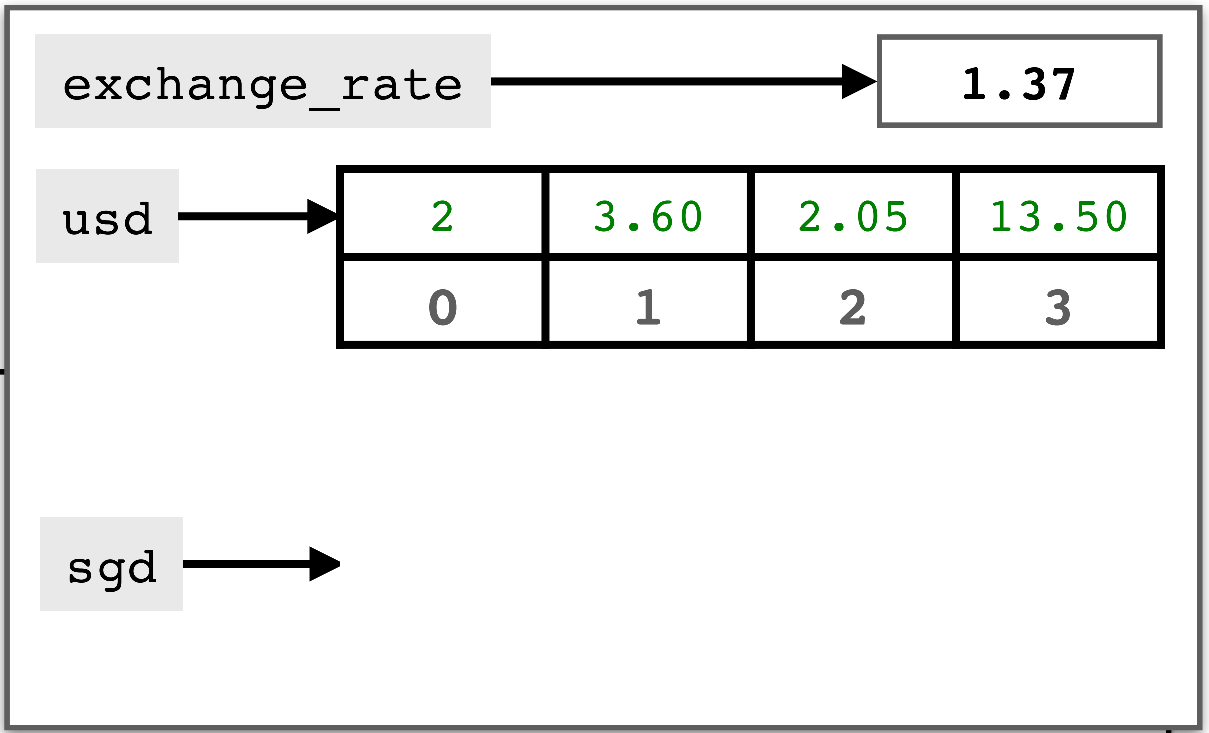
```
exchange_rate = 1.37
usd = [2, 3.60, 2.05, 13.50]

sgd = []
for trans in usd:
    sgd.append(trans*exchange_rate)

print(sgd)
```

```
[2.74, 4.932, 2.8085, 18.495]
```

Lists

- Lists as iterables
  - Iterating list items using a `for` loop

```
exchange_rate = 1.37
usd = [2, 3.60, 2.05, 13.50]

sgd = []
for trans in usd:
    sgd.append(trans*exchange_rate)

print(sgd)
```



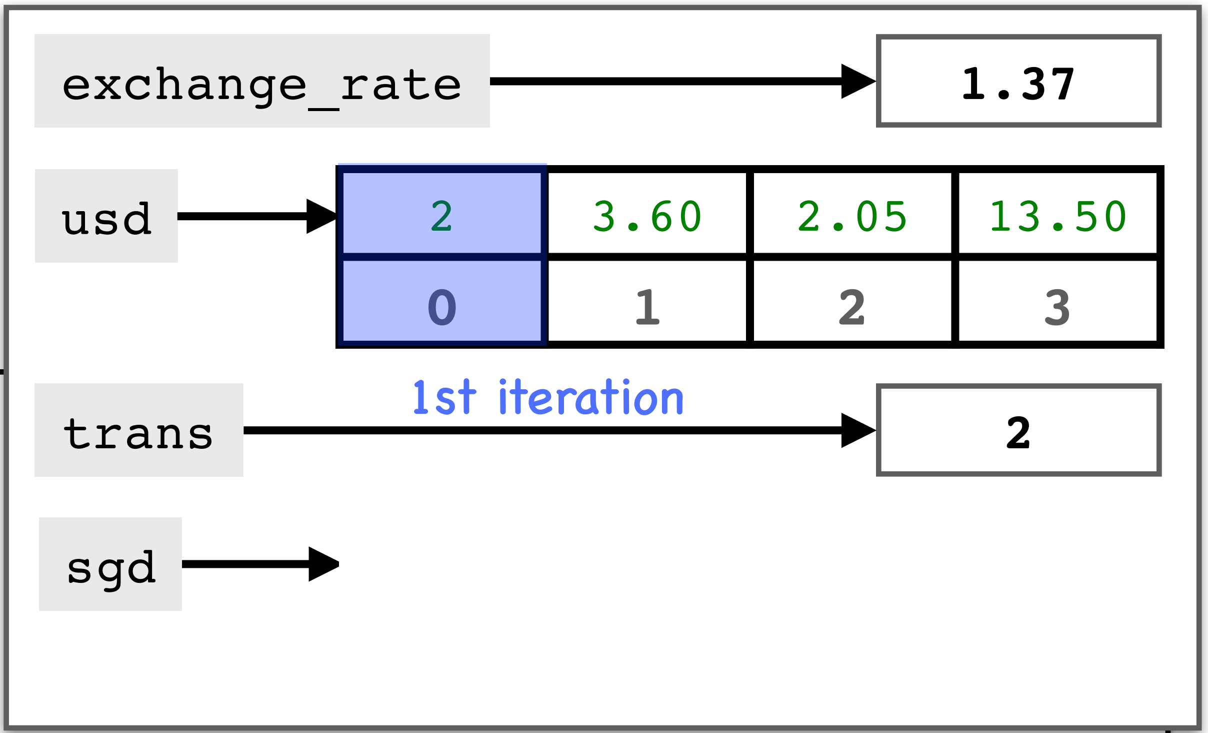
```
[2.74, 4.932, 2.8085, 18.495]
```

Lists

- Lists as iterables
  - Iterating list items using a `for` loop

```
exchange_rate = 1.37
usd = [2, 3.60, 2.05, 13.50]

sgd = []
for trans in usd:
    sgd.append(trans*exchange_rate)

print(sgd)
```

```
[2.74, 4.932, 2.8085, 18.495]
```

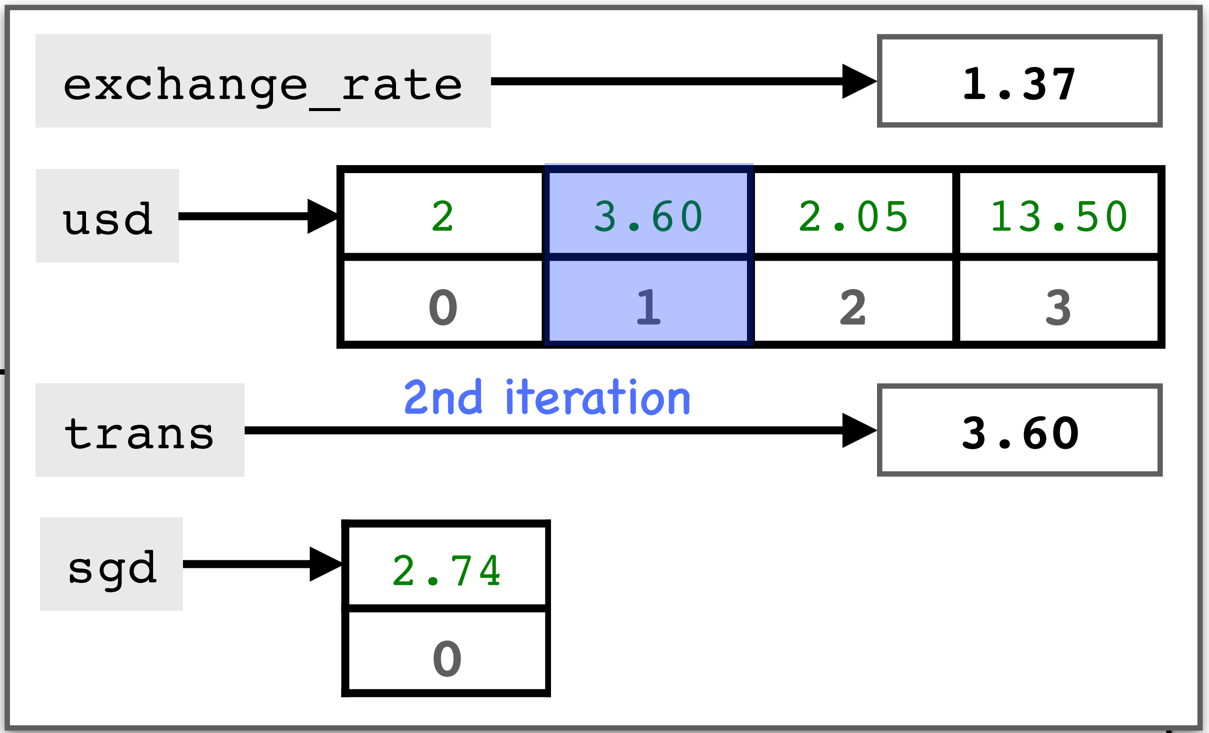


Lists

- Lists as iterables
  - Iterating list items using a `for` loop

```
exchange_rate = 1.37
usd = [2, 3.60, 2.05, 13.50]

sgd = []
for trans in usd:
    sgd.append(trans*exchange_rate)

print(sgd)
```

```
[2.74, 4.932, 2.8085, 18.495]
```

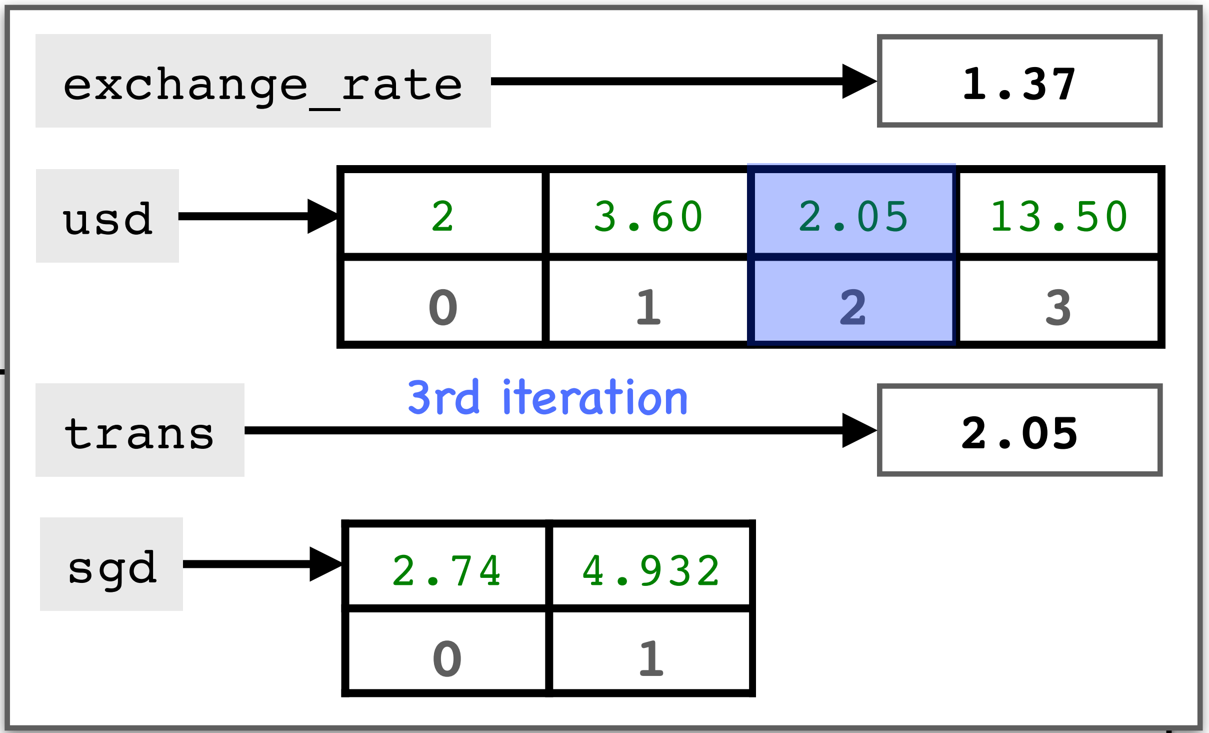


Lists

- Lists as iterables
  - Iterating list items using a `for` loop

```
exchange_rate = 1.37
usd = [2, 3.60, 2.05, 13.50]

sgd = []
for trans in usd:
    sgd.append(trans*exchange_rate)

print(sgd)
```

```
[2.74, 4.932, 2.8085, 18.495]
```

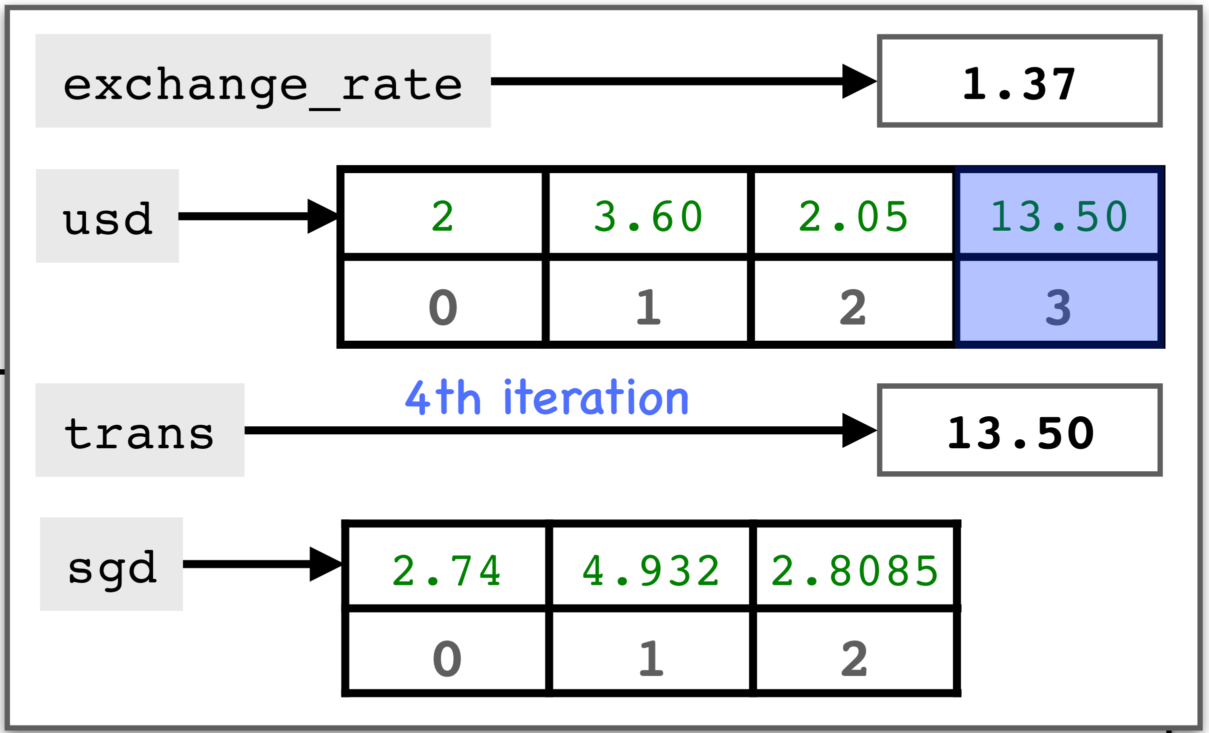


Lists

- Lists as iterables
  - Iterating list items using a `for` loop

```
exchange_rate = 1.37
usd = [2, 3.60, 2.05, 13.50]

sgd = []
for trans in usd:
    sgd.append(trans*exchange_rate)

print(sgd)
```

```
[2.74, 4.932, 2.8085, 18.495]
```

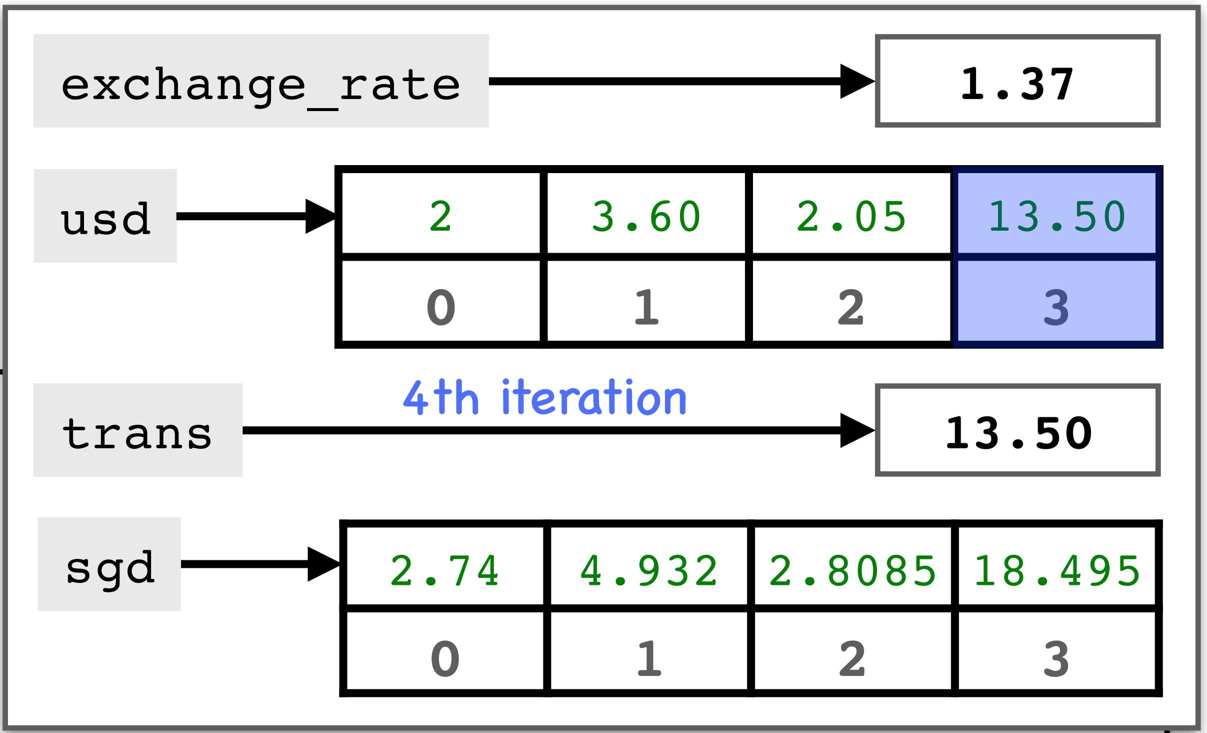


Lists

- Lists as iterables
  - Iterating list items using a `for` loop

```
exchange_rate = 1.37
usd = [2, 3.60, 2.05, 13.50]

sgd = []
for trans in usd:
    sgd.append(trans*exchange_rate)

print(sgd)
```

```
[2.74, 4.932, 2.8085, 18.495]
```



Lists

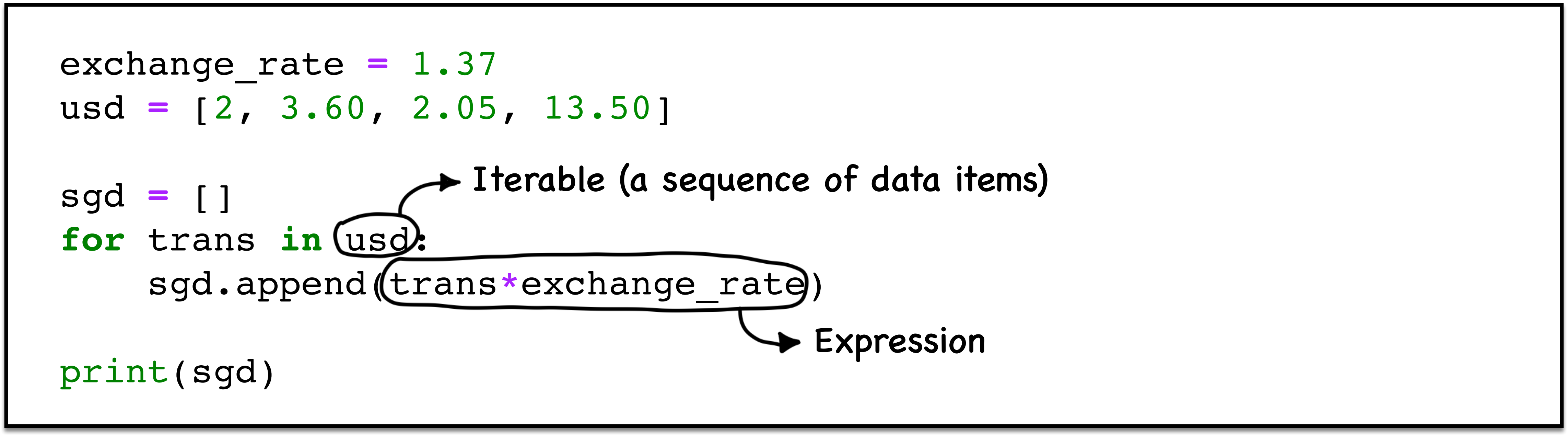
- Lists as iterables
    - Create a list using comprehensions

```
[expression for item in iterable]
```

Lists

- Lists as iterables
  - Create a list using comprehensions

```
[expression for item in iterable]
```



```
exchange_rate = 1.37
usd = [2, 3.60, 2.05, 13.50]

sgd = []
for trans in usd:
    sgd.append(trans*exchange_rate)

print(sgd)
```

```
[2.74, 4.932, 2.8085, 18.495]
```

Lists

- Lists as iterables
  - Create a list using comprehensions

```
[expression for item in iterable]
```

```
sgd = [trans*exchange_rate for trans in usd]
```

```
exchange_rate = 1.37
usd = [2, 3.60, 2.05, 13.50]

sgd = []
for trans in usd:
    sgd.append(trans*exchange_rate)

print(sgd)
```

Iterable (a sequence of data items)

Expression

```
[2.74, 4.932, 2.8085, 18.495]
```

Lists

- Lists as iterables
  - Create a list using comprehensions

```
[expression for item in iterable]
```

```
sgd = [trans*exchange_rate for trans in usd]
```

```
exchange_rate = 1.37
usd = [2, 3.60, 2.05, 13.50]

sgd = []
for trans in usd:
    sgd.append(trans*exchange_rate)

print(sgd)
```

Iterable (a sequence of data items)

Expression

```
[2.74, 4.932, 2.8085, 18.495]
```

Lists

- Lists as iterables
  - Create a list using comprehensions

```
[expression for item in iterable if conditions]
```

Lists

- Lists as iterables
    - Create a list using comprehensions

Example 3: Given a list of words, create a new list that includes all words starting with a vowel letter (A, E, I, O, or U).

```
words = ['AI', 'machine learning', 'analytics', 'prediction',
         'inference', 'regression', 'optimization']
```

Lists

- Lists as iterables
  - Create a list using comprehensions

Example 3: Given a list of words, create a new list that includes all words starting with a vowel letter (A, E, I, O, or U).

```
words = ['AI', 'machine learning', 'analytics', 'prediction',
         'inference', 'regression', ‘optimization']

new = []
for word in words:
    ...
    ...

print(new)
```

Lists

- Lists as iterables
    - Create a list using comprehensions

Example 3: Given a list of words, create a new list that includes all words starting with a vowel letter (A, E, I, O, or U).

```
words = ['AI', 'machine learning', 'analytics', 'prediction',
         'inference', 'regression', ‘optimization']

new = []
for word in words:
    if ... ... :
        new.append(word)

print(new)
```

Lists

- Lists as iterables
  - Create a list using comprehensions

Example 3: Given a list of words, create a new list that includes all words starting with a vowel letter (A, E, I, O, or U).

```
words = ['AI', 'machine learning', 'analytics', 'prediction',
         'inference', 'regression', 'optimization']

new = []
for word in words:
    if word[0].lower() in 'aeiou':
        new.append(word)

print(new)
```

```
['AI', 'analytics', 'inference', 'optimization']
```

Lists

- Lists as iterables
  - Create a list using comprehensions

```
[expression for item in iterable if conditions]
```

```
new = [word for word in words if word[0].lower() in 'aeiou']
```

```
words = ['AI', 'machine learning', 'analytics', 'prediction',
         'inference', 'regression', ‘optimization']

new = []
for word in words:
    if word[0].lower() in 'aeiou':
        new.append(word)

print(new)
```

```
['AI', 'analytics', 'inference', 'optimization']
```

Lists

- Lists as iterables
  - Create a list using comprehensions

```
[expression for item in iterable if conditions]
```

```
new = [word for word in words if word[0].lower() in 'aeiou']
```

```
words = ['AI', 'machine learning', 'analytics', 'prediction',
         'inference', 'regression', ‘optimization']

new = []
for word in words:
    if word[0].lower() in 'aeiou':
        new.append(word)

print(new)
```

Coding Style: List comprehension is preferred to a loop in creating new lists.

```
['AI', 'analytics', 'inference', 'optimization']
```