Working with strings

INTRODUCTION TO PYTHON FOR DEVELOPERS

George Boorman
Curriculum Manager, DataCamp

datacamp

Strings are everywhere!

- User profiles
- Search engines
- Large language models

[1] https://unsplash.com/@steve_j

Python knows single and double quotes

```
# This works
my_text = 'Hello, my name is George.'

# This also works
my_text = "Hello, my name is George."
```

- `'` = apostrophe = single quote

Advantages of double quotes

```
# Single quote string variable containing an apostrophe
my_text = 'Hello, my name's George.'
```

```
SyntaxError: invalid syntax.
```

```
# Double quote string variable containing an apostrophe
my_text = "Hello, my name's George."
print(my_text)
```

```
Hello, my name's George.
```

Methods

- Method = a function that is only available to a specific data type
- `str` methods

```
# Calling a string method
str_variable.method()
```

Replacing parts of strings

- `.replace(text_to_be_replaced, text_to_change_it_to)`

```
my_text = "Hello, my name's George."

# Replace George with John
my_text = my_text.replace("George", "John")
print(my_text)
```

```
Hello, my name's John.
```

- Common use cases
  - Reformatting e.g., change spaces to underscores
  - Fixing or removing typos

Changing case

```
current_top_album = "For All The Dogs"

# Convert to lowercase
current_top_album = current_top_album.lower()
print(current_top_album)
```

```
for all the dogs
```

```
# Change to upppercase
current_top_album = current_top_album.upper()
print(current_top_album)
```

```
FOR ALL THE DOGS
```

Sentences and paragraphs

```
nineteen_eightyfour = "It was a bright cold day in April, and the clocks were striking thirteen."
```

Multi-line strings

```
# Create a string variable over multiple lines
harry_potter = """Mr. and Mrs. Dursley,
of number four Privet Drive,
were proud to say that they were perfectly normal,
thank you very much.
"""
```

- `"""text"""` : Multi-line strings
  - Enhance readability
  - Avoid the need to use special characters
  - Longer text such as customer reviews
  - Used to describe what functions do

String cheat sheet

Syntax	Purpose	Example
Single quotes `''`	String variable	`my_string = 'My string'`
Double quotes `""`	String variable	`my_string = "My string"`
Triple quotes `""""""`	Multi-line string variable	`my_string = """My string"""`
`str.replace()`	Replacing parts of strings	`my_string = my_string.replace("text_to_remove", "text_to_change_to")`
`str.lower()`	Lowercase string	`my_string = my_string.lower()`
`str.upper()`	Uppercase string	`my_string = my_string.upper()`

[1] https://docs.python.org/3/library/stdtypes.html#string-methods

Let's practice!

INTRODUCTION TO PYTHON FOR DEVELOPERS

Lists

INTRODUCTION TO PYTHON FOR DEVELOPERS

George Boorman
Curriculum Manager, DataCamp

datacamp

Problem

```
# Variables of product prices
price_one = 10
price_two = 20
price_three = 30
price_four = 15
price_five = 25
price_six = 35
```

Lists to the rescue!

- List = store multiple values in a single variable
  - Can contain any combination of data types e.g., `str` , `float` , `int` , `bool`

```
# List of prices
prices = [10, 20, 30, 15, 25, 35]
```

```
# List of prices using variables as values
prices = [price_one, price_two, price_three,
          price_four, price_five, price_six]
```

Checking the data type

```
# Check the data type of a list
type(prices)
```

```
<class 'list'>
```

```
# Check the data type of a string
type("Hello")
```

```
<class 'str'>
```

Accessing elements of a list

```
# Print all values in the list variable
print(prices)
```

```
[10, 20, 30, 15, 25, 35]
```

- Lists are ordered
  - Can use subsetting or indexing
  - Python counts values starting from **zero** for the *first element*

Accessing elements of a list

- List = `[]`
- Subsetting = `a_list[index]`

```
prices = [10, 20, 30, 15, 25, 35]

# Get the value at the first index
prices[0]
```

```
10
```

```
# Get the value at the fourth index
prices[3]
```

```
15
```

Finding the last element in a list

```
prices = [10, 20, 30, 15, 25, 35]

# Get the last element of a list
prices[5]
```

```
35
```

```
# Get the last element of a list
prices[-1]
```

```
35
```

Accessing multiple elements

```
prices = [10, 20, 30, 15, 25, 35]

# Access the second and third elements
prices[1:3]
```

```
[20, 30]
```

- `[starting_element:last_element + 1]`
- Add one to the last element's index because:
  - Python returns everything up to **but not including** that index

Accessing multiple elements

```
# Access all elements from the fourth index onwards
prices[3:]
```

```
[15, 25, 35]
```

```
# Get the first three elements
prices[:3]
```

```
[10, 20, 30]
```

Alternating access

```
# Access every other element
prices[::2]
```

```
[10, 30, 25]
```

```
# Access every third element, starting at the second
prices[1::3]
```

```
[20, 25]
```

Lists cheat sheet

Syntax	Functionality
`a_list = [1, 2, 3, 4, 5, 6]`	Create a list variable
`a_list[0]`	Get the first element at index zero
`a_list[-1]`	Get the last element
`a_list[0:3]`	Get the first, second, and third elements
`a_list[:3]`	Get the first, second, and third elements
`a_list[2:]`	Get all elements from the third index onwards
`a_list[::2]`	Get every other element from the first index onwards

Let's practice!

INTRODUCTION TO PYTHON FOR DEVELOPERS

Dictionaries

INTRODUCTION TO PYTHON FOR DEVELOPERS

George Boorman
Curriculum Manager, DataCamp

datacamp

Products list

```
prices = [10, 20, 30, 15, 25, 35]

# Product IDs
product_ids = ["AG32", "HT91", "PL65", "OS31", "KB07", "TR48"]
```

Dictionary

- Dictionary = key-value pairs

Word / key

Definition / value

[1] https://unsplash.com/@sandym10

Why use dictionaries?

- `user_id`
- `order_number`
- `date`
- `value`
- `payment_method`

- `ip_address`
- `location`

[1] https://unsplash.com/@sharonmccutcheon; https://unsplash.com/@jjying

Creating a dictionary

```
# Creating a dictionary
products_dict =
```

Creating a dictionary

```
# Creating a dictionary
products_dict = {
```

Creating a dictionary

```
# Creating a dictionary
products_dict = {"AG32"
```

Creating a dictionary

```
# Creating a dictionary
products_dict = {"AG32":
```

Creating a dictionary

```
# Creating a dictionary
products_dict = {"AG32": 10
```

Creating a dictionary

```
# Creating a dictionary
products_dict = {"AG32": 10,
```

Creating a dictionary

```
# Creating a dictionary
products_dict = {"AG32": 10, "HT91": 20,
```

Creating a dictionary

```
# Creating a dictionary
products_dict = {"AG32": 10, "HT91": 20,
                 "PL65": 30, "OS31": 15,
                 "KB07": 25, "TR48": 35
```

Creating a dictionary

```
# Creating a dictionary
products_dict = {"AG32": 10, "HT91": 20,
                 "PL65": 30, "OS31": 15,
                 "KB07": 25, "TR48": 35}
```

Accessing a value based on the key

- Dictionaries are ordered
  - Allows values to be accessed by **subsetting on the key**
- What is the price of product ID `"AG32"` ?

```
# Find the product's price
products_dict["AG32"]
```

```
10
```

Accessing all values

```
# Get all values from a dictionary
products_dict.values()
```

```
dict_values([10, 20, 30, 15, 25, 35])
```

Accessing all keys

```
# Retrieve all keys in a dictionary
products_dict.keys()
```

```
dict_keys(['AG32', 'HT91', 'PL65', 'OS31', 'KB07', 'TR48'])
```

Viewing an entire dictionary

```
# Print the dictionary
print(products_dict)
```

```
{'AG32': 10, 'HT91': 20, 'PL65': 30, 'OS31': 15, 'KB07': 25, 'TR48': 35}
```

```
# Get all items (key-value pairs)
products_dict.items()
```

```
dict_items([('AG32', 10), ('HT91', 20), ('PL65', 30), ('OS31', 15), ('KB07', 25),
('TR48', 35)])
```

- `.items()` is useful when iterating or looping

Adding a key-value pair

```
# Add a new key-value pair
products_dict["UI56"] = 40
```

Updating a value

```
# Updating a value associated with an existing key
products_dict["HT91"] = 12
```

Duplicate keys

```
# Creating a dictionary with a duplicate key
products_dict = {"AG32": 10, "AG32": 20,
                 "PL65": 30, "OS31": 15,
                 "KB07": 25, "TR48": 35}

# Print the duplicate key's value
print(products_dict["AG32"])
```

```
20
```

Let's practice!

INTRODUCTION TO PYTHON FOR DEVELOPERS

Sets and tuples

INTRODUCTION TO PYTHON FOR DEVELOPERS

George Boorman
Curriculum Manager, DataCamp

datacamp

Sets

- Contain **unique** data
- Unchangeable*
  - Can add or remove values, but *cannot change them*
- Ideal to identify and remove duplicates
- Quick to search (compared to other data structures such as lists)

Creating a set

- Set = `{}`
- `:` = Dictionary
- No `:` = Set

```
# Create an attendee set
attendees_set = {"John Smith", "Alan Jones", "Roger Thompson",
                 "John Smith", "Brandon Sharp", "Sam Washington"}
print(attendees_set)
```

```
{'Alan Jones', 'Brandon Sharp', 'John Smith',
 'Roger Thompson', 'Sam Washington'}
```

Converting to a set

```
# Existing list variable
attendees_list = ["John Smith", "Alan Jones", "Roger Thompson",
                  "John Smith", "Brandon Sharp", "Sam Washington"]

# Convert to a set
attendees_set = set(attendees_list)

# Check the data type
type(attendees_set)
```

```
set
```

Converting to a set

```
print(attendees_set)
```

```
{'Sam Washington', 'Roger Thompson', 'Alan Jones', 'John Smith', 'Brandon Sharp'}
```

Limitations of sets

- Don't have an index
  - Can't have duplicates
  - Can't subset with `[]`

```
# Trying to subset a set
attendees_set[0]
```

```
TypeError: 'set' object is not subscriptable
```

Sorting a set

```
attendees_set = {"John Smith", "Alan Jones", "Roger Thompson",
                 "John Smith", "Brandon Sharp", "Sam Washington"}

# Sorting a set
sorted(attendees_set)
```

```
['Alan Jones', 'Brandon Sharp', 'John Smith', 'Roger Thompson', 'Sam Washington']
```

- `sorted()` returns a list

Tuples

- **Immutable** - *cannot be changed*
  - No adding values
  - No removing values
  - No changing values

- **Ordered**
  - Can subset by index i.e., `[0]`

[1] https://unsplash.com/@towfiqu999999

Creating a tuple

```
# Creating a tuple
office_locations = ("New York City", "London", "Leuven")

# Convert another data structure to a tuple
attendees = tuple(attendees_list)
```

Accessing tuples

```
# Access the second element
office_locations[1]
```

```
"London"
```

Data structures summary

Data structure	Syntax	Immutable	Allow duplicate values	Ordered	Subset with []
List	`[1, 2, 3]`	No	Yes	Yes	Yes - index
Dictionary	`{key:value}`	No	Yes	Yes	Yes - key
Set	`{1, 2, 3}`	No	No	No	No
Tuple	`(1, 2, 3)`	Yes	Yes	Yes	Yes - index

Let's practice!

INTRODUCTION TO PYTHON FOR DEVELOPERS