Working with strings

INTRODUCTION TO PYTHON FOR DEVELOPERS

Jasmin Ludolf
Senior Data Science Content Developer

datacamp

Strings are everywhere!

- Displaying messages
- Processing text input
- File names
- Formatting output
- Handling data

[1] Image generated by ChatGPT

Python knows single and double quotes

```
# This works
ingredient_name = 'San Marzano tomatoes'

# This also works
ingredient_name = "San Marzano tomatoes"
```

- `'` = apostrophe = single quote

Advantages of double quotes

```
# Single quote string variable containing an apostrophe
ingredient_name = 'Chef's special seasoning'
print(ingredient_name)
```

```
SyntaxError: invalid syntax.
```

```
# Double quote string variable containing an apostrophe
ingredient_name = "Chef's special seasoning"
print(ingredient_name)
```

```
Chef's special seasoning
```

Sentences and paragraphs

```
recipe_step = "Heat olive oil in a large pan and sauté garlic until fragrant"
```

Multi-line strings

```
# Create a string variable over multiple lines
recipe_instructions = """1. Bring a large pot of salted water to boil and cook pasta
2. Heat olive oil in a pan and sauté minced garlic until fragrant
3. Add chopped tomatoes and simmer for 10 minutes
4. Toss cooked pasta with tomato sauce and fresh basil leaves
"""
```

- `"""text"""` : Multi-line strings
  - Enhance readability
  - Used for documentation
  - Longer text such as instructions or error messages

Methods

- Method = a function that is only available to a specific data type
- `str` methods
  - Standardizing input, or transforming text

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

Replacing parts of strings

- `.replace(text_to_be_replaced, text_to_change_it_to)`

```
welcome_message = "Welcome to the recipe scaler, George"


welcome_message = welcome_message.replace("George", "John")
print(welcome_message)
```

```
Welcome to the recipe scaler, John
```

Changing case

- Standardize user inputs like emails

```
ingredient_name = "Basil Leaves"
# Convert to lowercase
ingredient_name = ingredient_name.lower()
print(ingredient_name)
```

```
basil leaves
```

```
# Change to uppercase
ingredient_name = ingredient_name.upper()
print(ingredient_name)
```

```
BASIL LEAVES
```

Let's practice!

INTRODUCTION TO PYTHON FOR DEVELOPERS

Lists

INTRODUCTION TO PYTHON FOR DEVELOPERS

Jasmin Ludolf
Senior Data Science Content Developer

datacamp

Problem

```
# Variables of ingredient names
ingredient_one = "pasta"
ingredient_two = "tomatoes"
ingredient_three = "garlic"
ingredient_four = "basil"
ingredient_five = "olive oil"
ingredient_six = "salt"
```

Lists to the rescue!

- List = store multiple values in a single variable
  - Can contain any combination of data types

```
# List of ingredients
ingredients = ["pasta", "tomatoes", "garlic", "basil", "olive oil", "salt"]
```

```
# List of ingredients using variables as values
ingredients = [ingredient_one, ingredient_two, ingredient_three,
               ingredient_four, ingredient_five, ingredient_six]
```

Checking the data type

```
# Check the data type of a list
print(type(ingredients))
```

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

Accessing elements of a list

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

```
['pasta', 'tomatoes', 'garlic', 'basil', 'olive oil', 'salt']
```

- Lists are ordered and indexed
  - Python assigns index starting from **zero** for the *first element*

Accessing elements of a list

- List = `[]`
- Access an element = `a_list[index]`

```
ingredients = ["pasta", "tomatoes",
"garlic", "basil", "olive oil", "salt"]

# Get the value at the first index
print(ingredients[0])
```

```
pasta
```

```
# Get the fourth element
print(ingredients[3])
```

```
basil
```

Finding the last element in a list

```
ingredients = ["pasta", "tomatoes", "garlic", "basil", "olive oil", "salt"]

# Get the last element of a list
print(ingredients[5])
```

```
salt
```

```
# Get the last element of a list
print(ingredients[-1])
```

```
salt
```

Accessing multiple elements

```
ingredients = ["pasta", "tomatoes", "garlic", "basil", "olive oil", "salt"]

# Access the second and third elements
print(ingredients[1:3])
```

```
["tomatoes", "garlic"]
```

- `[first_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

```
ingredients = ["pasta", "tomatoes", "garlic", "basil", "olive oil", "salt"]

# Access all elements from the third index onwards
print(ingredients[3:])
```

```
['basil', 'olive oil', 'salt']
```

```
# Get the first three elements
print(ingredients[:3])
```

```
['pasta', 'tomatoes', 'garlic']
```

Alternating access

```
ingredients = ["pasta", "tomatoes", "garlic", "basil", "olive oil", "salt"]
# Access every second element
print(ingredients[::2])
```

```
['pasta', 'garlic', 'olive oil']]
```

- Returns items at indexes zero, two, and four

```
# Access every third element, starting at the second
print(ingredients[1::3])
```

```
['tomatoes', 'olive oil']
```

- Returns items at indexes one and four

Let's practice!

INTRODUCTION TO PYTHON FOR DEVELOPERS

Dictionaries

INTRODUCTION TO PYTHON FOR DEVELOPERS

Jasmin Ludolf
Senior Data Science Content Developer

datacamp

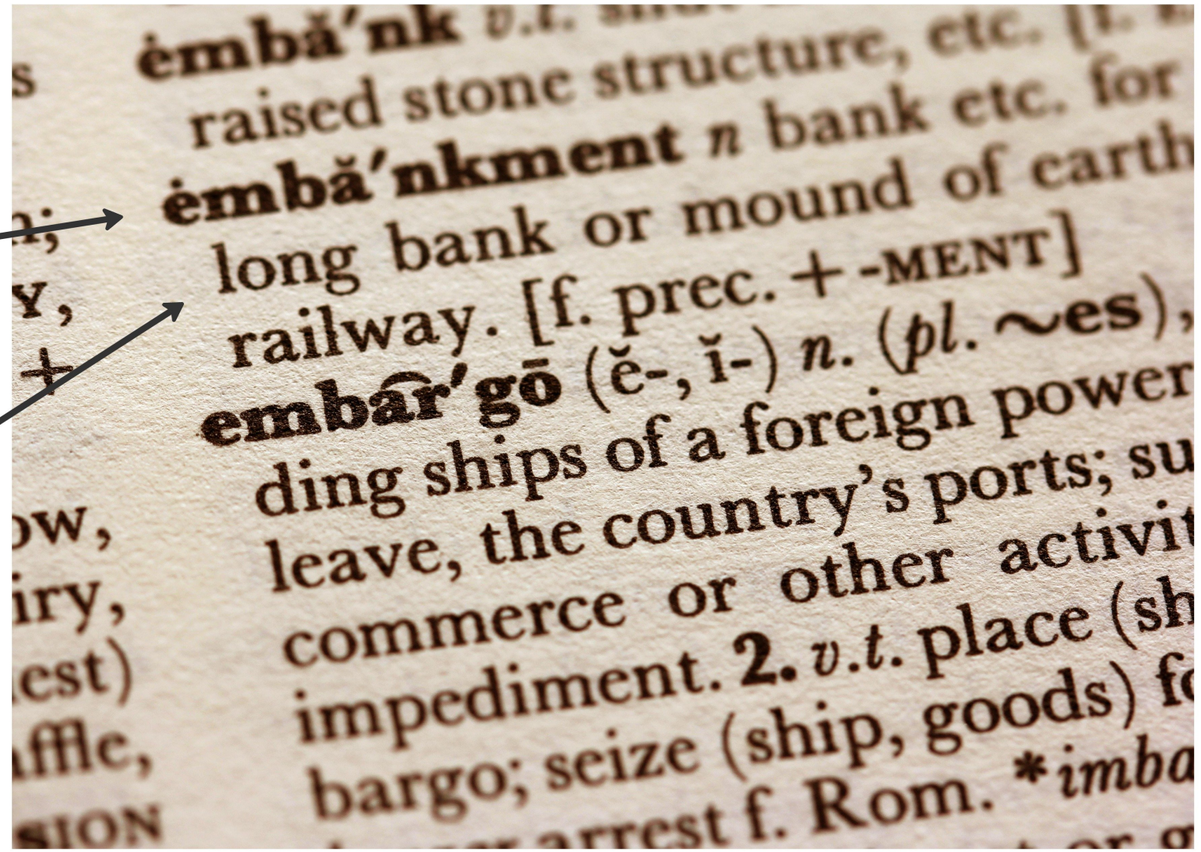
Ingredients list

```
ingredients = ["pasta", "tomatoes", "garlic", "basil", "olive oil", "salt"]

# Ingredient quantities (in grams)
quantities = [500, 400, 15, 20, 30, 5]
```

Dictionary

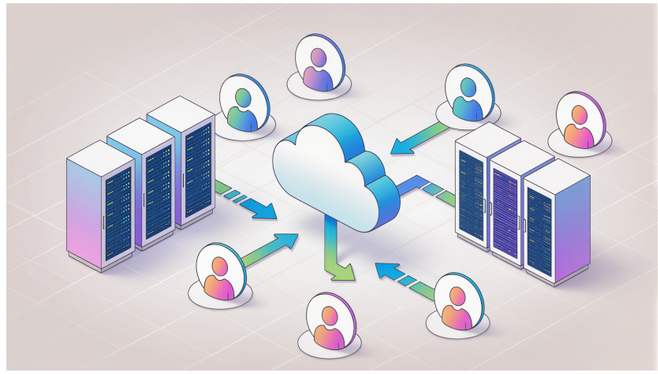
- Dictionary = key-value pairs



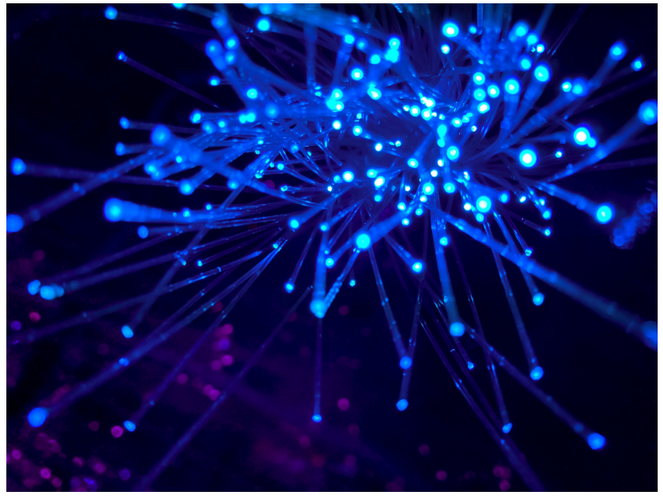
[1] https://unsplash.com/@sandym10

Why use dictionaries?

- `username`
- `email`
- `preferences`



- `ip_address`
- `location`



- Recipe scaler: link ingredients to quantities

[1] User image by Bilobaba Vladimir, Aviavlad; https://unsplash.com/@jjying

Creating a dictionary

```
# Creating a dictionary
recipe =
```

Creating a dictionary

```
# Creating a dictionary
recipe = {
```

Creating a dictionary

```
# Creating a dictionary
recipe = {"pasta"
```

Creating a dictionary

```
# Creating a dictionary
recipe = {"pasta":
```

Creating a dictionary

```
# Creating a dictionary
recipe = {"pasta": 500
```

Creating a dictionary

```
# Creating a dictionary
recipe = {"pasta": 500,
```

Creating a dictionary

```
# Creating a dictionary
recipe = {"pasta": 500,
          "tomatoes": 400,
```

Creating a dictionary

```
# Creating a dictionary
recipe = {"pasta": 500,
          "tomatoes": 400,
          "garlic": 15,
          "basil": 20,
          "olive oil": 30,
          "salt": 5
```

Creating a dictionary

```
# Creating a dictionary
recipe = {"pasta": 500,
          "tomatoes": 400,
          "garlic": 15,
          "basil": 20,
          "olive oil": 30,
          "salt": 5}
```

Accessing a value based on the key

- Dictionaries are ordered
  - Allows values to be accessed by **subsetting on the key**
- How much pasta do we need?

```
# Find the ingredient's quantity
print(recipe["pasta"])
```

```
500
```

Accessing all values

```
# Get all values from a dictionary
print(recipe.values())
```

```
dict_values([500, 400, 15, 20, 30, 5])
```

Accessing all keys

```
# Retrieve all keys in a dictionary
print(recipe.keys())
```

```
dict_keys(['pasta', 'tomatoes', 'garlic', 'basil', 'olive oil', 'salt'])
```

Viewing an entire dictionary

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

```
{'pasta': 500, 'tomatoes': 400, 'garlic': 15, 'basil': 20, 'olive oil': 30,
'salt': 5}
```

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

```
dict_items([('pasta', 500), ('tomatoes', 400), ('garlic', 15), ('basil', 20),
('olive oil', 30), ('salt', 5)])
```

Adding a key-value pair

```
# Add a new key-value pair
recipe["parmesan"] = 50
```

```
print(recipe)
```

```
{'pasta': 500, 'tomatoes': 400, 'garlic': 15, 'basil': 20, 'olive oil': 30,
'salt': 5, 'parmesan': 50}
```

Updating a value

```
print(recipe)
```

```
{'pasta': 500, 'tomatoes': 400, 'garlic': 15, 'basil': 20, 'olive oil': 30,
'salt': 5, 'parmesan': 50}
```

```
# Updating a value associated with an existing key
recipe["pasta"] = 1000
```

```
print(recipe)
```

```
{'pasta': 1000, 'tomatoes': 400, 'garlic': 15, 'basil': 20, 'olive oil': 30,
'salt': 5, 'parmesan': 50}
```

Duplicate keys

```
# Creating a dictionary with a duplicate key
recipe = {"pasta": 500, "garlic": 5,
          "garlic": 15, "basil": 20,
          "olive oil": 30, "salt": 5}

# Print the duplicate key's value
print(recipe["garlic"])
```

```
15
```

Let's practice!

INTRODUCTION TO PYTHON FOR DEVELOPERS

Sets and tuples

INTRODUCTION TO PYTHON FOR DEVELOPERS

Jasmin Ludolf
Senior Data Science Content Developer

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 a set of ingredients
ingredients = {"pasta", "tomatoes", "pasta",
               "basil", "garlic", "olive oil", "salt"}
print(ingredients)
```

```
{'pasta', 'tomatoes', 'garlic', 'basil', 'olive oil', 'salt'}
```

Converting to a set

```
# Existing list variable
ingredients_list = ["pasta", "tomatoes", "garlic", "basil"
                    "olive oil", "pasta", "salt"]

# Convert to a set
unique_ingredients = set(ingredients_list)

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

```
set
```

Converting to a set

```
print(unique_ingredients)
```

```
{'pasta', 'tomatoes', 'garlic', 'basil', 'olive oil'}
```

Limitations of sets

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

```
# Trying to subset a set
print(unique_ingredients[0])
```

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

Sorting a set

```
ingredients = {"pasta", "tomatoes", "garlic",
               "basil", "olive oil", "salt"}

# Sorting a set
print(sorted(ingredients))
```

```
['basil', 'garlic', 'olive oil', 'pasta', 'salt', 'tomatoes']
```

- `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]`

- Useful for location information or identifiers

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

Creating a tuple

```
# Creating a tuple
serving_sizes = (1, 2, 4, 6, 8)


# Convert another data structure to a tuple
ingredients_tuple = tuple(ingredients_list)
```

Accessing tuples

```
# A tuple
serving_sizes = (1, 2, 4, 6, 8)

# Access the second element
print(serving_size[1])
```

```
2
```

Let's practice!

INTRODUCTION TO PYTHON FOR DEVELOPERS