



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

ACCA5036

Scripting and Programming Concepts

Week 6: Data Structures in Python

Did you know?

Python was named after the British comedy group **Monty Python**, not the snake.

Trawsnewid Addysg; Trawsnewid Bywydau
Transforming Education; Transforming Lives



www.uwtsd.ac.uk

Objectives – This week

- Recap – datatypes (Quiz)
- Python built-in data structures- **Lists, Dictionaries, Tuples and Sets**
- -to understand how to efficiently organize, store, and manage data for effective problem-solving and program development.
- -recognizing Appropriate Use Cases



<https://forms.office.com/e/j5TZ3K8gNP>



Open Moodle in Lab computers

The screenshot shows a web browser window with the address bar displaying <https://moodle.uwtsd.ac.uk/course/view.php?id=12415§ion=7>. The browser tabs include 'Staff » Hub' and 'Tile: Week 4 : Looping | Scripting'. The Moodle interface features a dark blue header with the University of Wales Trinity Saint David logo and navigation links: 'Dashboard', 'My modules', 'Moodle archives', and 'Useful links'. On the right side of the header, there are notification icons, a user profile 'SS', and an 'Edit mode' toggle. The main content area lists course materials for 'Week 4 : Looping Recording':

- Exercise : Looping DOCX
- Looping Solutions DOCX
- Validation using Loops
- SS Week 4 Loops (While and For)
- week 4 handouts PDF
- Activities- Looping
- Activities - Looping : Solution

A lock icon and the text 'Available from 3 July 2025, 10:00 AM' are at the bottom of the list. A right-hand sidebar contains two sections: 'Upcoming events' (stating there are no events) and 'Staff Moodle Training Survey' (with a prompt to click an image to take the survey). The survey image shows a form titled 'Moodle Training Survey' with a text area and a submit button.

Open Thonny IDE in your laptops/lab computers



Recap – datatypes (Quiz)

What is the data type of the value True in Python?

- A) Int
- B) Str
- C) Bool
- D) float

Recap – datatypes (Quiz)

What is the data type of the value True in Python?

A) Int

B) Str

C) Bool

D) float

Recap – datatypes (Quiz)

What is the output of `type(3.5)` in Python?

- A) `<class 'float'>`
- B) `<class 'int'>`
- C) `<class 'double'>`
- D) `<class 'decimal'>`

Recap – datatypes (Quiz)

What is the output of `type(3.5)` in Python?

- A) <class 'float'>
- B) <class 'int'>
- C) <class 'double'>
- D) <class 'decimal'>

Recap – datatypes (Quiz)

What is the output of: `type("123")`?

- A) Int
- B) float
- C) str
- D) bool

Recap – datatypes (Quiz)

What is the output of: `type("123")`?

- A) `Int`
- B) `float`
- C) `str`
- D) `bool`

Recap – datatypes (Quiz)

Which of the following statements is true regarding the basic datatypes like int and str?

- A. They consume too much memory
- B. They cannot hold multiple values together
- C. They are not supported in Python

Recap – datatypes (Quiz)

Which of the following statements is true regarding the basic datatypes like int and str?

- A. They consume too much memory
- B. They cannot hold multiple values together
- C. They are not supported in Python

Recap – datatypes (Quiz)

What happens if you want to store 100 student names using only basic variables like str?

A. You need to create 100 separate variables

B. You can store them using the int type

C. It automatically stores in memory efficiently

Storing 5 names using str

```
student1 = "Alice"  
student2 = "Bob"  
student3 = "Charlie"  
student4 = "David"  
student5 = "Eva"  
print(student1)  
print(student2)  
print(student3)  
print(student4)  
print(student5)
```

Storing 5 names using Lists

```
student_names = ["Alice", "Bob", "Charlie", "David",  
"Eva"]  
print(student_names)
```

Compare

```
student1 = "Alice"  
student2 = "Bob"  
student3 = "Charlie"  
student4 = "David"  
student5 = "Eva"  
print(student1)  
print(student2)  
print(student3)  
print(student4)  
print(student5)
```

```
student_names = ["Alice", "Bob", "Charlie", "David", "Eva"]  
print(student_names)
```

Compare

```
student1 = "Alice"  
student2 = "Bob"  
student3 = "Charlie"  
student4 = "David"  
student5 = "Eva"  
print(student1)  
print(student2)  
print(student3)  
print(student4)  
print(student5)
```

Which is better to store 100 student names using only individual str variables?

```
student_names = ["Alice", "Bob", "Charlie", "David", "Eva"]  
print(student_names)
```

Compare

```
student1 = "Alice"  
student2 = "Bob"  
student3 = "Charlie"  
student4 = "David"  
student5 = "Eva"  
print(student1)  
print(student2)  
print(student3)  
print(student4)  
print(student5)
```

Which is better to store 100 student names using only individual str variables?

7,900 bytes

6,764 bytes

```
student_names = ["Alice", "Bob", "Charlie", "David", "Eva"]  
print(student_names)
```

Why Prefer Data Structures

Data Structures

Scalability

Why Prefer Data Structures

Maintainability

Data Structures

Scalability

Why Prefer Data Structures

Code Reusability

Maintainability

Data Structures

Scalability

Why Prefer Data Structures

Code Reusability

Data Structures

Scalability

Code Readability

Maintainability

Data structures in Python

Data structures in Python are **containers** that organize, manage, and store data **efficiently** so it can be accessed and modified easily.

Python provides both **built-in data structures** and allows for the creation of **custom ones**.

Data structures in Python

- **List:** Ordered, mutable collection
(e.g., `[1, 2, 3]`)
- **Tuple:** Ordered, immutable collection
(e.g., `(1, 2, 3)`)
- **Set:** Unordered, mutable collection with unique elements
(e.g., `{1, 2, 3}`)
- **Dictionary:** Key-value pairs
(e.g., `{'key': 'value'}`)

List Datatype

- a **list** is a built-in data structure
- to store multiple items in a single variable
- can hold different types of data, including numbers, strings, and even other lists
- They are ordered, mutable (changeable), and allow duplicate elements.
- You can create a list by placing values inside square brackets [], separated by commas.

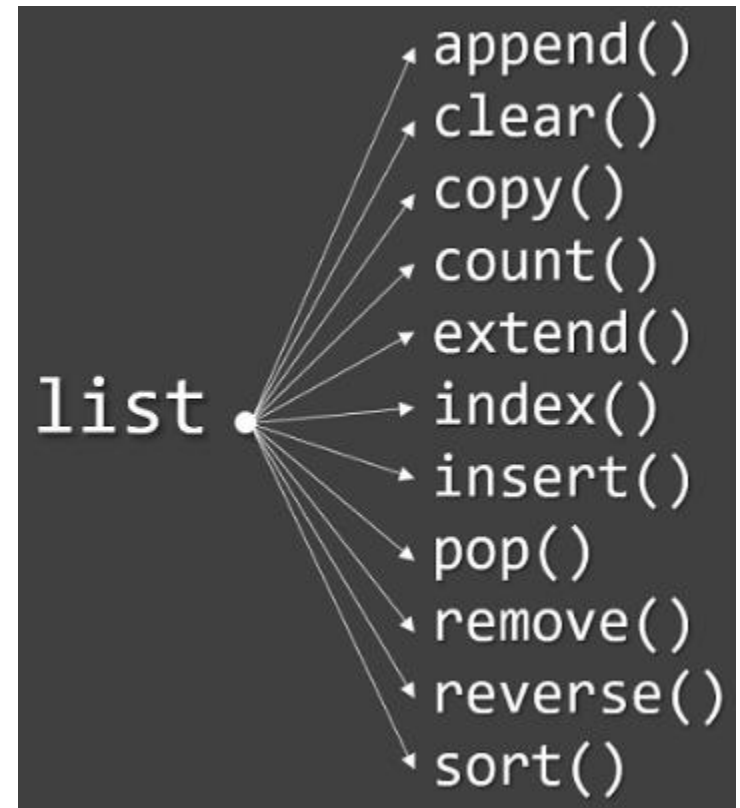
Built in Methods for List

- Every list automatically has the following methods.

e.g.,

`mylist.append()`

`mylist.sort()`



Creating a List

```
my_list = [1, 2, 3, 4, 5]  
print(my_list)
```

```
my_list = [5, 2, 3, 4, 5]  
print(my_list)
```

Creating a List

```
my_list = ["apple", "banana", "cherry"]  
print(my_list)
```

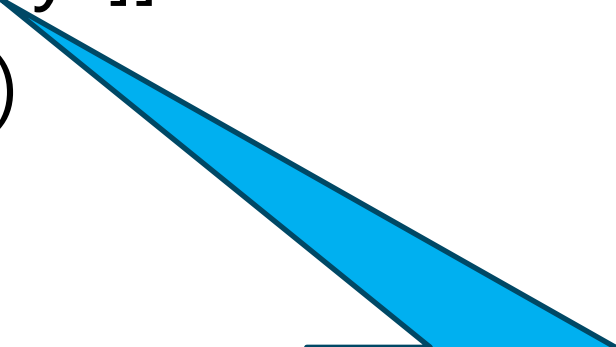
```
mixed_list = [1, "hello", 3.14, True]  
print(mixed_list)
```

Creating a List

```
mixed_list = [1, "hello", 3.14, True, ["apple",  
"banana", "cherry"]]  
print(mixed_list)
```

Creating a List

```
mixed_list = [1, "hello", 3.14, True, ["apple",  
"banana", "cherry"]]  
print(mixed_list)
```



List with in the list

List Operations

Accessing Elements

```
fruits = ["apple", "banana", "cherry"]  
print(fruits[0])
```

```
fruits = ["apple", "banana", "cherry"]  
print(fruits[-1])
```

List Operations

Accessing Elements

```
fruits = ["apple", "banana", "cherry"]  
print(fruits[4])
```

```
fruits = ["apple", "banana", "cherry"]  
print(fruits[-4])
```

List Operations

List Slicing

```
fruits = ["apple", "banana", "cherry"]  
print(fruits[1:2])
```

```
numbers = [1, 2, 3, 4, 5]  
print(numbers[1:4])
```

List Operations

Modifying a List

```
fruits = ["apple", "banana", "cherry"]  
fruits[1] = "blueberry"  
print(fruits)
```

```
fruits = ["apple", "banana", "cherry"]  
fruits.append("dates")  
print(fruits)
```

List Operations

Modifying a List

```
fruits = ["apple", "banana", "cherry"]  
fruits.insert(1, "blueberry")  
print(fruits)
```

```
fruits = ["apple", "banana", "cherry"]  
fruits.extend(["date", "elderberry"])  
print(fruits)
```

List Operations

Modifying a List

```
fruits = ["apple", "banana", "cherry"]  
fruits.remove("banana")  
print(fruits)
```

```
fruits = ["apple", "banana", "cherry"]  
popped_item = fruits.pop(1)  
print(fruits)  
print("Popped item:", popped_item)
```

```
fruits = ["apple", "banana", "cherry"]  
del fruits[1]
```

Knowledge Check - List

- Q1: Which of the following correctly initializes a list in Python?
 - A. `list = {1, 2, 3}`
 - B. `list = [1, 2, 3]`
 - C. `list = (1, 2, 3)`
 - D. `list = <1, 2, 3>`

Knowledge Check - List

- Q2: What is the output of the following code?

```
my_list = [10, 20, 30, 40]  
print(my_list[2])
```

- A) 10
- B) 20
- C) 30
- D) 40

Knowledge Check - List

- Q3: What is the output of the following code?

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)
```

- A) [1, 2, 3]
- B) [1, 2, 3, 4]
- C) [4, 1, 2, 3]
- D) [1, 2, 4]

Knowledge Check - List

- Q3: What is the output of the following code?

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)
```

- A) [1, 2, 3]
- B) [1, 2, 3, 4]
- C) [4, 1, 2, 3]
- D) [1, 2, 4]

Knowledge Check - List

- Q4: What is the output of the following code?

```
my_list = [5, 10, 15, 20]  
my_list[1] = 50  
print(my_list)
```

- A) [5, 10, 15, 20]
- B) [50, 10, 15, 20]
- C) [5, 50, 15, 20]
- D) [5, 10, 50, 20]

Knowledge Check - List

- Q4: What is the output of the following code?

```
my_list = [5, 10, 15, 20]  
my_list[1] = 50  
print(my_list)
```

- A) [5, 10, 15, 20]
- B) [50, 10, 15, 20]
- C) [5, 50, 15, 20]
- D) [5, 10, 50, 20]

Knowledge Check - List

- Q5: What is the output of the following code?

```
my_list = [5, 10, 15, 20]
my_list[1] = 50
print(my_list)
my_list.insert(1, 15)
print(my_list)
```

- A) [5, 50, 15, 20]
[5, 15, 15, 20]
- B) [5, 50, 15, 20]
[5, 15, 50, 15, 20]

Knowledge Check - List

- Q5: What is the output of the following code?

```
my_list = [5, 10, 15, 20]
my_list[1] = 50
print(my_list)
my_list.insert(1, 15)
print(my_list)
```

- A) [5, 50, 15, 20]
 [5, 15, 15, 20]
- B) [5, 50, 15, 20]
 [5, 15, 50, 15, 20]

Knowledge Check - List

- Q6: Which one is not a valid List below?

A. numbers = [1, 2, 3, 4]

B. mixed = [1, "apple", 3.14, [5, 6]]

C. empty = []

D. None of the above

Knowledge Check - List

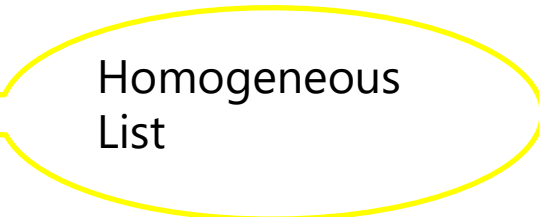
- Q6: Which one is not a valid List below?

A. numbers = [1, 2, 3, 4]

B. mixed = [1, "apple", 3.14, [5, 6]]

C. empty = []

D. None of the above



Homogeneous
List

Knowledge Check - List

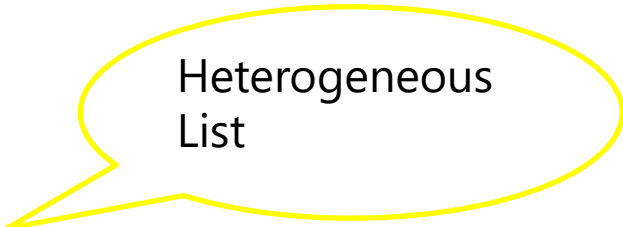
- Q6: Which one is not a valid List below?

A. numbers = [1, 2, 3, 4]

B. mixed = [1, "apple", 3.14, [5, 6]]

C. empty = []

D. None of the above



Heterogeneous
List

Knowledge Check - List

- Q6: Which one is not a valid List below?

A. numbers = [1, 2, 3, 4]

B. mixed = [1, "apple", 3.14, [5, 6]]

C. empty = []

D. None of the above

Empty List

Knowledge Check - List

- Q7: What is the output of this code?

```
my_list = [3, 1, 4, 1, 5]
```

```
my_list.sort()
```

```
print(my_list)
```

A) [3, 1, 4, 1, 5]

B) [1, 1, 3, 4, 5]

C) [5, 4, 3, 1, 1]

D) [1, 3, 1, 4, 5]

Knowledge Check - List

- Q7: What is the output of this code?

```
my_list = [3, 1, 4, 1, 5]
```

```
my_list.sort()
```

```
print(my_list)
```

A) [3, 1, 4, 1, 5]

B) [1, 1, 3, 4, 5]

C) [5, 4, 3, 1, 1]

D) [1, 3, 1, 4, 5]

Knowledge Check - List

- Q8: What is the output of this code?

```
my_list = [1, "apple", 3.14, [5, 6]]
```

```
my_list.sort()
```

```
print(my_list)
```

Knowledge Check - List

- Q8: What is the output of this code?

```
my_list = [1, "apple", 3.14, [5, 6]]
```

```
my_list.sort()
```

```
print(my_list)
```

TypeError: '<' not supported between instances of 'str' and 'int'

Knowledge Check - List

- Q8: What is the output of this code?

```
my_list = [1, 2, 3, 4, 5]  
print(len(my_list))
```

- A) 4
- B) 5
- C) 6
- D) Error

Knowledge Check - List

- Q8: What is the output of this code?

```
my_list = [1, 2, 3, 4, 5]
```

```
print(len(my_list))
```

A) 4

B) 5

C) 6

D) Error

Knowledge Check - List

- Q10: What is the output of this code?

```
my_list = [1, "apple", 3.14, [5, 6]]  
print(len(my_list))
```

- A) Error
- B) 4
- C) 5
- D) 3

Knowledge Check - List

- Q10: What is the output of this code?

```
my_list = [1, "apple", 3.14, [5, 6]]  
print(len(my_list))
```

A) Error

B) 4

C) 5

D) 3

Print Elements of a List

```
fruits = ["apple", "banana", "cherry"]
```

```
for fruit in fruits:  
    print(fruit)
```

Print List Elements with Index

```
colors = ["red", "green", "blue"]
```

```
for index, color in enumerate(colors):  
    print(f"{index}: {color}")
```

Print List Elements with Index

```
colors = ["red", "green", "blue"]
```

```
for index, color in enumerate(colors):  
    print(f"{index}: {color}")
```

Print List Elements with custom Index

```
fruits = ["apple", "banana", "cherry"]
```

```
for index, fruit in enumerate(fruits, start=1):  
    print(index, fruit)
```

Using enumerate() with a list comprehension

```
fruits = ["apple", "banana", "cherry"]
```

```
indexed_fruits = [(index, fruit) for index, fruit in enumerate(fruits)]  
print(indexed_fruits)
```

Using enumerate() to modify a list

```
fruits = ["apple", "banana", "cherry"]
```

```
for i, fruit in enumerate(fruits):
```

```
    fruits[i] = fruit.upper()
```

```
print(fruits)
```

To-do list creator

The user enters tasks one by one.

When they type "done", the program stops taking input.

It then displays the tasks with a numbered list.

To-do list creator

```
tasks = []
```

```
while True:
```

```
    task = input("Enter a task (or 'done' to finish): ")
```

```
    if task.lower() == "done":
```

```
        break
```

```
    tasks.append(task)
```

```
print("\nYour To-Do List:")
```

```
for index, task in enumerate(tasks, start=1):
```

```
    print(f"{index}. {task}")
```

Simple Quiz App

The program asks three questions.

The user enters their answers.

It checks correctness and keeps track of the score.

Finally, it prints the total score.

Simple Quiz App

```
questions = [  
    ("What is the capital of France?", "Paris"),  
    ("What is 2 + 2?", "4"),  
    ("What is the color of the sky?", "Blue"),  
]  
  
score = 0  
  
for index, (question, answer) in enumerate(questions, start=1):  
    user_answer = input(f"Q{index}: {question} ").strip()  
    if user_answer.lower() == answer.lower():  
        print("Correct!\n")  
        score += 1  
    else:  
        print(f"Wrong! The correct answer is {answer}.\n")  
  
print(f"You got {score}/{len(questions)} correct!")
```

Shopping Cart System

The user enters items for their shopping cart.

It displays the list with numbers.

The user can remove an item by entering its number.

The updated cart is displayed

Shopping Cart System

```
cart = []
```

```
while True:
```

```
    item = input("Enter an item to add to the cart (or 'done' to finish): ")
```

```
    if item.lower() == "done":
```

```
        break
```

```
    cart.append(item)
```

```
print("\nYour Shopping Cart:")
```

```
for index, item in enumerate(cart, start=1):
```

```
    print(f"{index}. {item}")
```

```
# Removing an item
```

```
remove_index = int(input("\nEnter item number to remove (or 0 to skip): "))
```

```
if remove_index > 0 and remove_index <= len(cart):
```

```
    cart.pop(remove_index - 1)
```

```
print("\nUpdated Cart:")
```

```
for index, item in enumerate(cart, start=1):
```

```
    print(f"{index}. {item}")
```

Data structure – list Recap & Processing in python

1. To initialize a list:

```
list = []
```

This will set the list to empty

2. To process a list:

```
for i in list:
```

This will deal with all of the values in the list

3. To check contents of a list if i in list:

This will test if the value i is in the list

if i not in list:

This will test if the value i is not in the list

4. To get the length of a list:

```
len(list)
```

This will return the length of a list

```
if len(list) == 0
```

This will test to see if the list is empty

5. To append to a list:

```
list.append(i)
```

This will append the item i to the list

6. to access the last element of a list

```
my_list[-1]
```

<https://forms.office.com/e/0K5Q89qK2q>



Next week

- More data structures
- Sets
- Dictionaries

Thank you

Gracias

Merci

Danke

Grazie

Obrigado / Obrigada

Dank je / Dank u

Спасибо (Spasibo)

谢谢 (Xièxiè)

ありがとう (Arigatou)

감사합니다 (Gamsahamnida)

धन्यवाद (Dhanyavaad)

شكرا (Shukran)

Teşekkürler

Ευχαριστώ (Efharisto)

תודה (Toda)

ขอบคุณ (Khxbkhun)

Asante

Salamat

using **sets** in python....

What is the output of the following code?

```
my_set = {1, 2, 3, 2, 1}  
print(my_set)
```

- A) {1, 2, 3, 2, 1}
- B) {1, 2, 3}
- C) [1, 2, 3]
- D) Error

using **sets** in python....

What is the output of the following code?

```
my_set = {1, 2, 3, 2, 1}  
print(my_set)
```

A) {1, 2, 3, 2, 1}

B) {1, 2, 3}

C) [1, 2, 3]

D) Error

using **sets** in python....

What is the output of the following code?

```
my_set = {1, "apple", 3.14, (1, 2)}  
print(my_set)
```

- A) {3.14, 1, (1, 2), 'apple'}
- B) {1, 3.14, (1,2)}
- C) [1, (1,2)]
- D) Error

using **sets** in python....

What is the output of the following code?

```
my_set = {1, "apple", 3.14, (1, 2)}  
print(my_set)
```

- A) {3.14, 1, (1, 2), 'apple'}
- B) {1, 3.14, (1,2)}
- C) [1, (1,2)]
- D) Error

using **sets** in python....

Which method adds an element to a set?

```
my_set = {1, 2, 3}
```

What goes here to add 4 to the set?

- A) my_set.add(4)
- B) my_set.append(4)
- C) my_set.insert(4)
- D) my_set.push(4)

using **sets** in python....

Which method adds an element to a set?

```
my_set = {1, 2, 3}
```

What goes here to add 4 to the set?

- A) my_set.add(4)
- B) my_set.append(4)
- C) my_set.insert(4)
- D) my_set.push(4)

using **sets** in python....

What is the output of this code?

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 & set2)
```

- A) {1, 2}
- B) {3}
- C) {1, 2, 3, 4, 5}
- D) Error

using **sets** in python....

What is the output of this code?

```
set1 = {1, 2, 3}  
set2 = {3, 4, 5}  
print(set1 & set2)
```

A) {1, 2}

B) {3}

C) {1, 2, 3, 4, 5}

D) Error

using **sets** in python....

What happens when you execute this code?

```
my_set = {1, 2, 3}  
my_set.remove(4)
```

- A) Removes 4 from the set.
- B) Does nothing.
- C) Raises a KeyError.
- D) Adds 4 to the set.

using **sets** in python....

What happens when you execute this code?

```
my_set = {1, 2, 3}  
my_set.remove(4)
```

- A) Removes 4 from the set.
- B) Does nothing.
- C) Raises a KeyError.
- D) Adds 4 to the set.

using **sets** in python....

```
set1 = {1, 2, 3}  
set2 = {3, 4, 5}  
print(set1 | set2)
```

{1, 2, 3, 4, 5}

```
my_set = {5, 10, 15}  
if 10 in my_set:
```

#complete the code

The in operator checks whether 10 is a member of the set my_set.

using **sets** in python....

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 | set2)
```

{1, 2, 3, 4, 5}

```
my_set = {5, 10, 15}
if 10 in my_set:
    print("10 is in the set")
else:
    print("10 is not in the
set")
```

The in operator checks whether 10 is a member of the set my_set.

using **sets** in python....

Write a python program to test if a string value is contained in a set, if it is in the set return true otherwise false.

Example:

```
colours = {"red", "yellow", "blue"}
```

```
x = "red"
```

Should return **true** as red is in the set.

using **sets** in python....

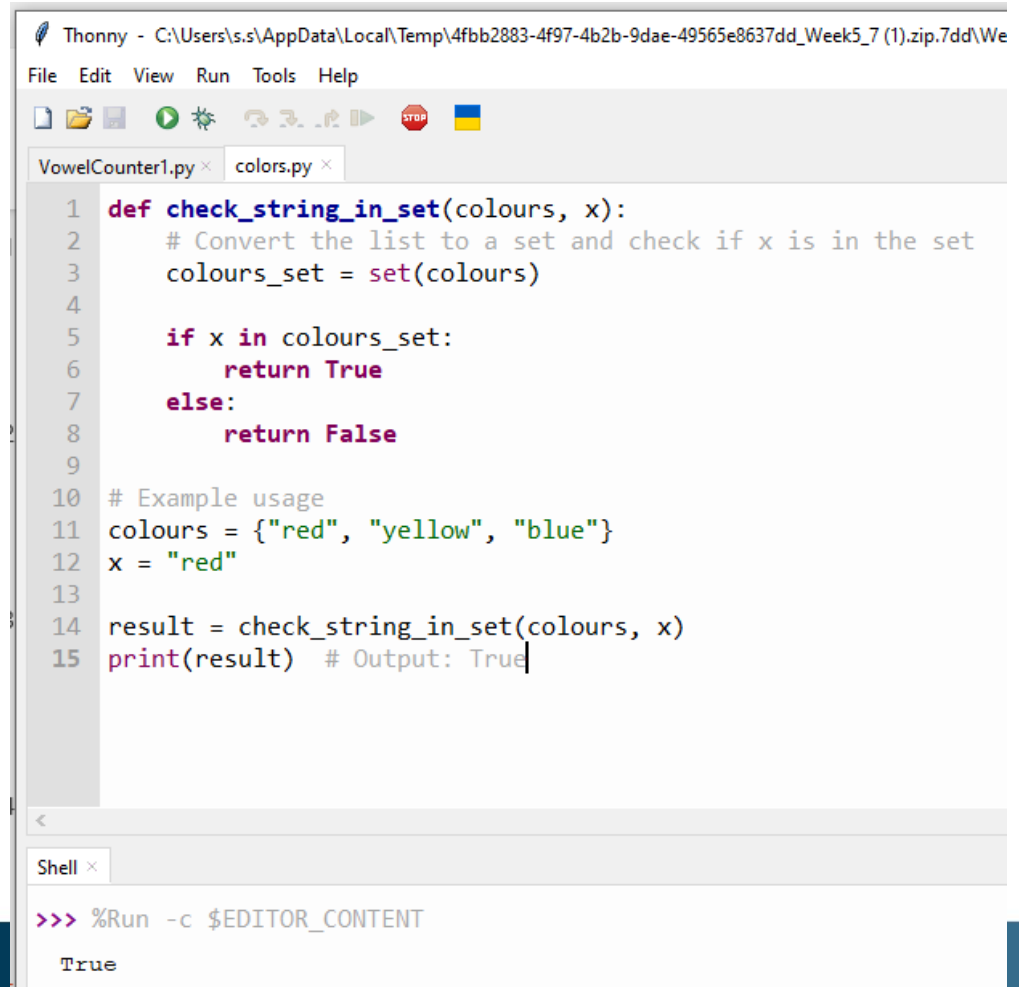
Write a python program to test if a string value is contained in a set, if it is in the set return true otherwise false.

Example:

`colours = {"red", "yellow", "blue"}`

`x = "red"`

Should return **true** as red is in the set.



```
Thonny - C:\Users\s.s\AppData\Local\Temp\4fbb2883-4f97-4b2b-9dae-49565e8637dd_Week5_7 (1).zip.7dd\We
File Edit View Run Tools Help
VowelCounter1.py x colors.py x
1 def check_string_in_set(colours, x):
2     # Convert the list to a set and check if x is in the set
3     colours_set = set(colours)
4
5     if x in colours_set:
6         return True
7     else:
8         return False
9
10 # Example usage
11 colours = {"red", "yellow", "blue"}
12 x = "red"
13
14 result = check_string_in_set(colours, x)
15 print(result) # Output: True

Shell x
>>> %Run -c $EDITOR_CONTENT
True
```

using **sets** in python....

Write a python program to test if a string value is contained in a set, if it is in the set return true otherwise false.

Example:

```
colours = {"red", "yellow", "blue"}  
x = "red"
```

Should return **true** as red is in the set.

using Dictionary in python....

Which of the following is the correct way to create a dictionary in Python?

- a) `dict = (1, 2, 3)`
- b) `dict = { 'name': 'John', 'age': 25 }`
- c) `dict = [ ('name', 'John'), ('age', 25) ]`
- d) `dict = ('name', 'John', 'age', 25)`

using Dictionary in python....

How do you access the value associated with the key 'age' in the dictionary

```
person = {'name': 'Alice', 'age': 30, 'city': 'New York'}?
```

- a) `person['age']`
- b) `person.age`
- c) `person.get('age')`
- d) Both a and c

using Dictionary in python....

What will be the output of the following code?

```
person = {'name': 'Alice', 'age': 30, 'city': 'New York'}
```

```
person['age'] = 35
```

```
print(person)
```

- a) {'name': 'Alice', 'age': 30, 'city': 'New York'}
- b) {'name': 'Alice', 'age': 35, 'city': 'New York'}
- c) {'name': 'Alice', 'age': 30, 'city': 'Los Angeles'}
- d) {'name': 'Alice', 'age': 30}

using Dictionary in python....

How do you add a new key-value pair email: 'alice@example.com' to the dictionary person?

- a) `person['email'] = 'alice@example.com'`
- b) `person.add('email', 'alice@example.com')`
- c) `person.append('email', 'alice@example.com')`
- d) `person.insert('email', 'alice@example.com')`

using Dictionary in python....

What does the del keyword do when used with a dictionary, as in `del person['age']`?

- a) It deletes the entire dictionary.
- b) It deletes the key 'age' and its associated value.
- c) It deletes only the value associated with 'age'.
- d) It raises a `KeyError` if the key doesn't exist.

using Dictionary in python....

Which of the following expressions checks if the key 'city' exists in the dictionary person?

- a) 'city' in person
- b) person.has_key('city')
- c) person['city'] is not None
- d) person.city is not None

using Dictionary in python....

What will `person.get('phone', 'N/A')` return if the dictionary

`person = {'name': 'Alice', 'age': 30, 'city': 'New York'}` does not contain the key 'phone'?

- a) None
- b) 'N/A'
- c) KeyError
- d) 'phone'

using Dictionary in python....

What does the method `person.items()` return?

- a) A list of the dictionary's keys.
- b) A list of the dictionary's values.
- c) A list of key-value pairs as tuples.
- d) A list of keys and their corresponding data types.

using Dictionary in python....

Which of the following is a valid dictionary in Python?

- a) {'name': 'Alice', 'age': 30, 'is_student': True}
- b) {'name': 'Alice', 30: 'age', 'is_student': True}
- c) {'Alice', 30, 'is_student'}
- d) {'name': 'Alice', 'age': 30, [1,2,3]: 'numbers'}

using Dictionary in python....

How can you merge two dictionaries, dict1 = {'a': 1, 'b': 2} and dict2 = {'c': 3, 'd': 4}?

- a) dict1.merge(dict2)
- b) dict1.update(dict2)
- c) dict1 + dict2
- d) dict1.append(dict2)

Summary

Key Points:

- Loops are essential for repeating actions in code.
- *While loops* continue as long as a condition is true.
- *For loops* iterate over a sequence of items.
- *Nested loops* are useful for multi-dimensional problems.
- Loop control statements (*break*, *continue*) alter the flow of loops.