Day 1

Variables behave like boxes. You can store different data types in them such as strings and lists. Tip: Pick names that make sense when creating variables.

```
user_prompt = "Enter a todo:"

todo1 = input(user_prompt)
todo2 = input(user_prompt)
todo3 = input(user_prompt)

todos = [todo1, todo2, todo3, "Hello"]

print(todos)
print(type(todo1))
```

Arguments are inputs used by functions. They can be any data type such as a string, list, etc. They can also be a variable associated with a data type.

Strings are types to represent text in your programs. They start and end with double or single quotes.

Functions are entities designed to perform a particular action. For example, a print function prints out text in the command line. An input function also prints out text in the command line but it also lets the user enter data.

Lists are types to represent a series of items. The items can be any other type such as string, numbers, and even lists.

© Python Mega Course

udemy.com/the-python-mega-course

Day 2

While loops are code blocks that execute code over and over again.

```
user_prompt = "Enter a todo:"

todos = []

while True:
    todo = input(user_prompt)
    print(todo.capitalize())
    todos.append(todo)
```

Methods perform actions in relation to the objects they are attached to. In this example, the **append** method appends items to the list object. The **capitalize** method capitalizes the first letter of its string object.

Indentation is white space that comes before code lines. Indented lines indicate they belong to the unindented line above them. In this example, the three indented lines belong to the unindented **while**-loop line.

Four white spaces are recommended for indentation.

© Python Mega Course

udemy.com/the-python-mega-course

Day 3

Match-case is a code block that allows the programmer to match a given value against a series of other values. Match-case is useful when you expect the user may enter different predefined values such as certain commands, days of the week, etc.

```
todos = []

while True:
    user_action = input("Type add, show, or exit:")
    user_action = user_action.strip()

    match user_action:
        case 'add':
            todo = input("Enter a todo: ")
            todos.append(todo)
        case 'show':
            for item in todos:
                print(item)
        case 'exit':
            break

print("Bye!")
```

for-loops are blocks of code that iterate over objects that are made of other objects (e.g., lists). The loop will perform the same operation over all the items of the list.

break is a statement. It is an optional part of a while loop. If the interpreter reaches the break statement, the while-loop will stop executing.

! Anything that is outside the while-loop will only be executed if the while-loop stops executing.

© Python Mega Course

udemy.com/the-python-mega-course

Day 4

Converting between datatypes can be done with functions such as *int*, *float*, and *str*.

```
case 'edit':
    number = int(input("Number of the todo to edit: "))
    number = number - 1
    new_todo = input("Enter new todo: ")
    todos[number] = new_todo
```

It is possible to update a variable you have defined earlier. All you have to do is redefine the variable using a new value or an expression that produces a new value (i.e., *number - 1).*

Accessing list items can be done using the *list[x]* syntax where x is the index of the item you want to access. Keep in mind that the indexing system starts from zero (i.e., the first item of the list has an index of zero).

List items can be replaced using the syntax *list[x] = new_item* where x is the index of the item you want to replace and *new_item* is the value of the new item.

© Python Mega Course

udemy.com/the-python-mega-course

Day 5

Enumerate creates an object with the structure:
[(index, item), (index, item), (index, item)]
That kind of object makes it possible to iterate using two variables.

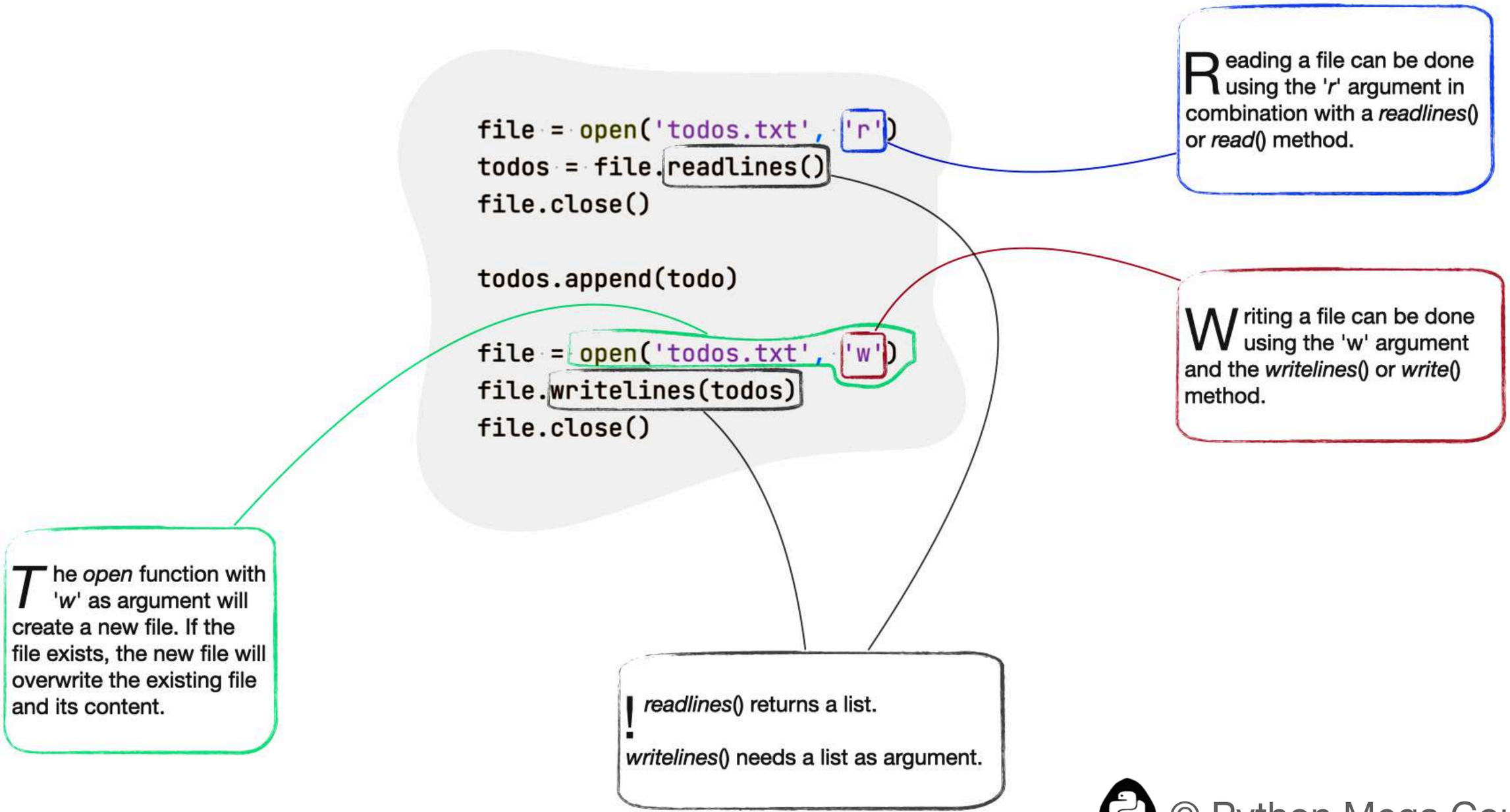
```
for index, item in enumerate(todos):
    row = f"{index + 1}-{item}"
    print(row)
```

f-strings construct strings. They are specialized to replace the *{variable}* part with the value of the variable. It is also possible to have an expressions inside {} and not only variables.

© Python Mega Course

udemy.com/the-python-mega-course

Day 6



© Python Mega Course

udemy.com/the-python-mega-course

Day 7

List comprehensions can create a new list by modifying an existing list.

1 The first half of the comprehension contains the expression that creates the new items.

2 The second half contains the for-syntax.

```
new_todos = [item.strip('\n') for item in todos]
```

3 The new list will be stored in this variable.

4 This variable represents each item of the existing list.

5 This variable holds the existing list whose items are used to create new items for the new list.

© Python Mega Course

udemy.com/the-python-mega-course

Day 8

The with-context manager is a helper to better handle file reading and writing.

```
with open('todos-delete.txt', 'r') as file:
    todos = file.readlines()

with open('todos-delete.txt', 'w') as file:
    file.writelines(todos)
```

1 The file will be closed implicitly once all the indented lines have been executed.

2 This is the variable that will hold the file object. It is the equivalent of ***file = open('todos-delete.txt', 'w')***

© Python Mega Course

udemy.com/the-python-mega-course

Day 9

If-elif-else can be used to check multiple conditions. Conditions are expressions that evaluate to either True or False.

1 The interpreter always checks the conditions of the if-line first.

```
if 'add' in user_action:
    todo = user_action[4:]

elif 'show' in user_action:
    with open('todos-delete.txt', 'r') as file:
        todos = file.readlines()

else:
    print("Command is not valid.")
```

2 If the condition of the if-line **does not** evaluate to True, the interpreter goes to check the condition of the elif-line.

3 If all conditions under the if-lines and the elif-lines are False, the interpreter executes the code indented under the else-line.

© Python Mega Course

udemy.com/the-python-mega-course

Day 10

Try-except can be used when you want to anticipate a specific kind of error. You can use it to display a friendly error message to the user instead of letting the program end abruptly showing the user an error message that they can hardly understand.

1 The interpreter will first try to execute the code indented under "try".

```
try:
    number = int(user_action[9:])

    with open('todos-delete.txt', 'r') as file:
        todos = file.readlines()
    index = number - 1
    todo_to_remove = todos[index].strip('\n')
    todos.pop(index)

    with open('todos-delete.txt', 'w') as file:
        file.writelines(todos)

    message = f"Todo {todo_to_remove} was removed from the list."
    print(message)
except IndexError:
    print("There is no item with that number.")
    continue
```

2 If the code under "try" has the error anticipated in the except-line, the code indented under "except" will be executed.

3 If you don't specify the error which you anticipate, then, if the code under "try" has ANY type of error, the code indented under "except" will be executed.

The recommendation is that you should specify the error as we are doing here with the IndexError.

© Python Mega Course

udemy.com/the-python-mega-course

Day 11

Custom functions are instructions we write to describe a certain process.

1 The instructions that describe the process are written in the **function definition.**

In this specific example, the instructions are:
1. Open the "todos-delete.txt" file
2. Extract text from the text file and store the text in a list.
3. Return the list as the output of the function.

```
def get_todos():
    with open('todos-delete.txt', 'r') as file_local:
        todos_local = file_local.readlines()
    return todos_local

...
...
todos = get_todos()
```

2 The instructions are executed by calling the function.

3 The value returned by executing the instructions can be stored in a variable.

© Python Mega Course

udemy.com/the-python-mega-course

Day 12

```python
def get_todos(filepath):
    with open(filepath, 'r') as file_local:
        todos_local = file_local.readlines()
    return todos_local


def write_todos(filepath, todos_arg):
    with open(filepath, 'w') as file:
        file.writelines(todos_arg)

todos = get_todos("todos.txt")
```

Arguments are also known as parameters. They are local variables that get a value dynamically when the function is called.

Argument values are assigned to arguments when the functions is called.

© Python Mega Course

udemy.com/the-python-mega-course

Day 13

Default arguments are arguments which are given a value in the function definition.

Non-default argument should be listed first (i.e., *todos_arg*), then the default arguments (i.e., *filepath*).

Doc strings are triple-quote strings defined in the function definition. They are shown as help documentation when *help(function)* is used.

```
def write_todos(todos_arg, filepath="todos.txt"):
    """ Write the to-do items list in the text file."""
    with open(filepath, 'w') as file:
        file.writelines(todos_arg)


write_todos(todos)
```

Default arguments don't have to be included in the function call unless you want to change their default value.

© Python Mega Course

udemy.com/the-python-mega-course

Day 14

Code can be kept more organized it is distributed across different Python files. Usually, function definitions are placed in one file, and the frontend interface in another. This is especially useful for larger programs.

```
from functions import get_todos, write_todos

todos = get_todos()
```

One Python file can be imported into another Python file using a "**from *module* import *object***" syntax.

The imported function can be called in the file where it is imported.

```
import functions

todos = functions.get_todos()
```

Another way to import a file is to use the "**import module**" syntax.

Once the module is imported through the "import module" method, the functions of that module can be called by referencing the module first.

© Python Mega Course

udemy.com/the-python-mega-course

Day 15

Standard modules are files containing functions that do more special operations than the functions included in the global Python namespace.

```
import functions
import time

now = time.strftime("%b %d, %Y %H:%M:%S")
print("It is", now)
```

Standard modules can be imported using an import statement just like local modules are imported.

Standard module functions can be called using the "**module.function()**" syntax.

© Python Mega Course

udemy.com/the-python-mega-course

Day 16

1 Third-party libraries such as PySimpleGUI are collections of Python **functions** and **types** which we can call and instantiate in our own programs. To be able to use such objets, we need to install the library with "pip install library" and then import it.

```
import PySimpleGUI as sg

label = sg.Text("Type in a to-do")
input_box = sg.InputText(tooltip="Enter todo")
add_button = sg.Button("Add")

window = sg.Window('My To-Do App', layout=[[label], [input_box, add_button]])
window.read()
print("Hello")
window.close()
```

2 Once the library is imported, you need to refer to the library name or its variable representation (i.e., sg) to be able to use the library functions or types.

3 Libraries can offer many types. For example, the PySimpleGUI library offers a Window type, Text type, InputText type, Button type, etc. We use such types to create Window, Text, InputText, Button and other instances. You can create as many instances as you want.

4 Once an object instance is stored in a variable, the methods of that type of instance can be accessed (e.g., read() and close()).

© Python Mega Course

udemy.com/the-python-mega-course

Day 17

1 The while loop serves as a listener. It helps you listen for events happening in the program and watch the values of the widgets as they change.

```
while True:
    event, values = window.read()
    match event:
        case "Add":
            todos = functions.get_todos()
            new_todo = values['todo'] + "\n"
            todos.append(new_todo)
            functions.write_todos(todos)
            window['todos'].update(values=todos)
```

2 The **event** variable holds the name or the key of the widget that was just clicked. The widget can be a button or some other clickable element.

3 The **values** variable holds a dictionary. The dictionary contains the current values entered or selected by the user in the widgets.

4 We can try to match the value of the **event** variable and perform different actions depending on what the current value of that variable is.

5 Widgets can be accessed with **window['widget_key']** and their values can be updated using the **update()** method.

© Python Mega Course

udemy.com/the-python-mega-course

Day 18

1 The while loop runs whenever an event happens unless you supply a timeout value. In that case the loop will run continuously every X (i.e., 200) milliseconds.

```
while True:
    event, values = window.read(timeout=200)
    window["clock"].update(value=time.strftime("%b %d, %Y %H:%M:%S"))

    except IndexError:
        sg.popup("Please select an item first.", font=("Helvetica", 20))

    case "Exit":
        break
    case 'todos':
        window['todo'].update(value=values['todos'][0])
    case sg.WIN_CLOSED:
        break
```

2 Whenever you want to update the value of a widget constantly, you can use the **update()** method in conjunction with the **timeout** argument explained in step 1.

3 If you expect an error, you should handle it with a try-except block and display a message to the user using a popup window.

4 The close-program functionality can be implemented both by handling the behaviour of the X button and also by adding an *Exit* button.

When the user presses one of those buttons we can break the while-loop with a ***break*** statement. Breaking the while-loop also breaks the program.

© Python Mega Course

udemy.com/the-python-mega-course

Day 19

```python
import streamlit as st
import functions

todos = functions.get_todos()

def add_todo():
    todo = st.session_state["new_todo"] + "\n"
    todos.append(todo)
    functions.write_todos(todos)


st.title("My Todo App")
st.subheader("This is my todo app.")
st.write("This app is to increase your productivity.")


for index, todo in enumerate(todos):
    checkbox = st.checkbox(todo, key=todo)
    if checkbox:
        todos.pop(index)
        functions.write_todos(todos)
        del st.session_state[todo]
        st.experimental_rerun()

st.text_input(label="", placeholder="Add new todo...",
              on_change=add_todo, key='new_todo')
```

Streamlit is a web framework library for Python. It offers methods that produce webpage objects such as **titles**, **headers** and simple **paragraphs**.

Streamlit also provides more interactive objects such as **checkboxes** and **text_input**.

© Python Mega Course

udemy.com/the-python-mega-course