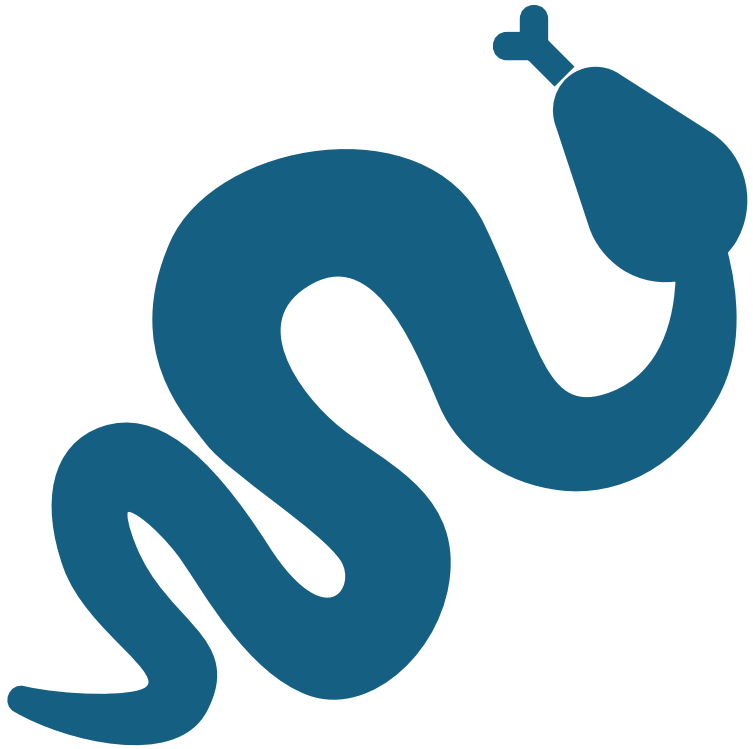


Lesson 5

# Python Fundamental

# Revisit

List

Slicing a List

Methods to Manipulate a List: append, copy, clear, insert, pop, remove, sort

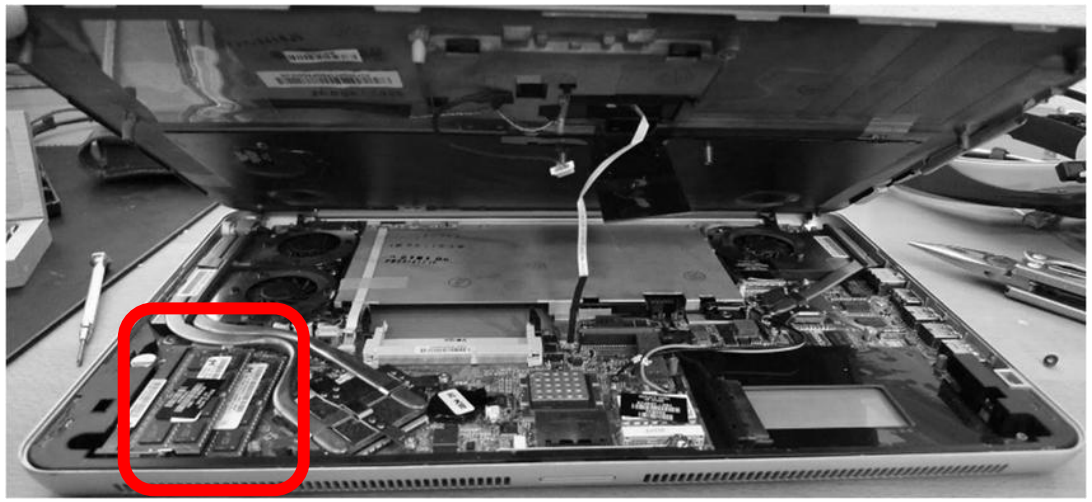
Exercise: Shopping List

# Technique We've Learned

- Random Library
- Decision Making: if-elif-else
- Logical Operators (==, !=, <, <=, >, >=)
- Boolean Operators (and, or, not)
- Boolean Values (True / False)
- Functions: range(start, stop, step)
- Control Structure: for loop, while loop
- Data Type: List

# Data Store (Short Term)

- Variables (integer, float, string, list) are store in memory
- Once computer is power off, variables are gone

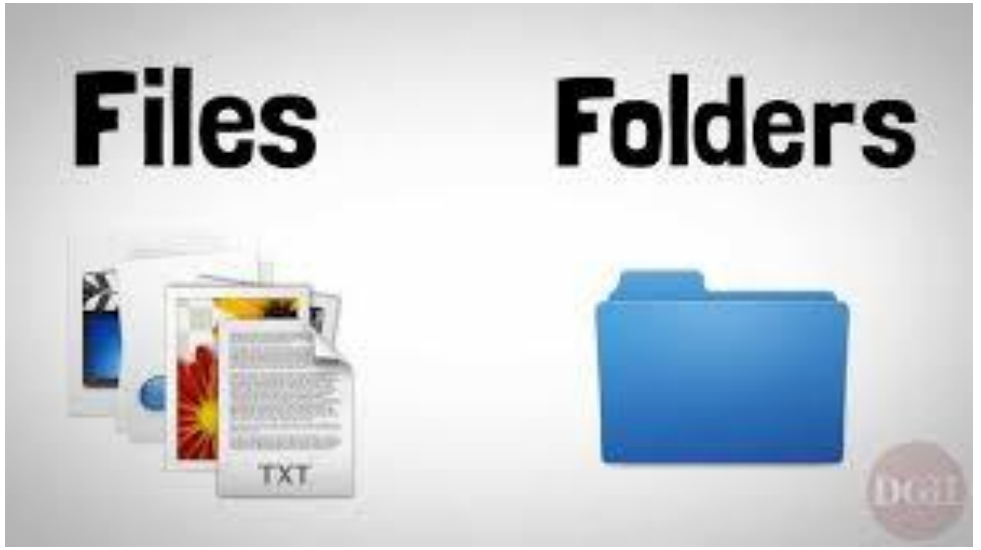


# Data Store (Long Term)

- Transfer memory data to files


Write

Read



# Python Text File Handling

| Mode | Operations | Behavior if file exist | Behavior if file doesn't exist |
|---|---|---|---|
| r | Read file content | Read file content | Error raise: FileNotFoundError |
| x | Create content | Error raise: FileExistsError | Create the file |
| w | Write content | Write file content | Create the file |
| a | Appen content | Append content to the file | Create the file |

# Open and Close a File

- Below code illustrate the basic operation of opening a text file call 'filedemo.txt' as write mode 'w'
- The open function return a file handler. The returned handler is store in variable 'f'
- After processing the file, use close() function to close the. Close is necessary because sometimes content is store in buffer. Perform Close function ensure content flash out from buffer and write to file.
- Code:

```
f = open('filedemo.txt', 'w')
f.write("I'm learning Python programming !\n")
f.write("This is lesson 5 !\n")
f.close()
```

# A Better Approach

```
with open('filedemo.txt', 'w') as f:
    f.write("I'm learning Python programming !\n")
    f.write("This is lesson 5 !\n")
```

- The **`with open('filedemo.txt', 'w') as f:`** statement automatically manages the file close operation **`f.close()`**.
- Once the **`with`** block is ends, Python automatically closes the file, even if an error occurs

# Scope of Variable

- Code line #4 generates error
- The variable 'f' defined inside a with statement is only valid within that block (line 2 and line 3 indented code)
- Variable f is outside the with block for line 4

```
1 with open('filedemo.txt', 'w') as f:
2     f.write("I'm learning Python programming !\n")
3     f.write("This is lesson 5 !\n")
4 f.write("This line will error")
```

# File Write Operations

```
#file write operation
with open('filedemo.txt', 'w') as f:
    f.write("I'm learning Python programming !\n")
    f.write("This is lesson 5!\n")

print('File filedemo.txt has been written')
```

# File Append Operations

```
#file append operation

with open('filedemo.txt', 'a') as f:
    f.write("Appended line\n")

print("A new line has added to filedemo.txt")
```

# File Read Operation

```
# setup a variable 'file'

with open('filedemo.txt', 'r') as f:
    # loop each line of the file
    for l in f:
        # loop each line of the file
        print(l, end='')
```

# Lab

- Enhance Shopping Cart on last lesson:
  - Use while loop to ask for shopping items (instead of using for loop to execute 5 times)
  - If user hit enter without input any shopping items:
    - Sort the shopping list
    - Save shopping list item to a text file
    - End the program
    - Print a message inform user the shopping list has saved to 'file name'

# Functions

**A reusable code block perform specific task**

**4 elements to form a function:**

1. Define the function: **def**
2. Input **parameter**
3. Run the **task**
4. Return values

```
# define the function
def sum_up(a, b):
    sum = a * b
    return sum
# reuse sum_up function above
print(sum_up(3, 5))
print(sum_up(2, 4))
print(sum_up(8, 12))
```

# Design a Function

- Each function should perform one specific task clearly defined by its name.

**Good Example**

```
def calculate_area(length, width):
    return length * width

def display_result(area):
    print(f"The area is: {area}")

def save_to_file(data, filename):
    with open(filename, 'w') as f:
        f.write(data)
```

**Poor Example**

```
def calculate_and_display_and_save(length,
        width, filename):
    area = length * width  # calculating
    print(f"Area: {area}")  # displaying
    with open(filename, 'w') as f:  # saving
        f.write(str(area))
    return area
```

# Function - Signature (Parameters)

- Function signature defines what inputs a function expects.
- (Input) Parameter is optional
- Number of required parameters could be defined as:
  - Required
  - Required with default value
  - Multiple parameters
  - Variable number of parameters
  - Keyword arguments

```
# Required parameters
def greet(name):
    return f"Hello, {name}!"

# Default parameters
def greet_with_title(name, title="Mr."):
    return f"Hello, {title} {name}!"

# Multiple parameters
def calculate_rectangle_area(length, width):
    return length * width

# Variable number of parameters (*args)
def sum_numbers(*numbers):
    return sum(numbers)

# Keyword arguments (**kwargs)
def create_profile(**details):
    return details
```

# Function – Return Values

- Return value hand over calculated result to the caller.
- (Output) Return value is optional
- Return could be:
  - Single return value
  - Multiple return values
  - Different data types
- If not return value, use **`pass`** statement to indicate the end of function block

```
# Single return value
def get_full_name(first_name, last_name):
    return f"{first_name} {last_name}"

result = get_full_name("John", "Doe")
print(result)  # Output: John Doe

# Multiple return values
def get_name_parts(full_name):
    parts = full_name.split()
    first_name = parts[0]
    last_name = parts[-1]
    return first_name, last_name

# Return Different Data Types
def analyze_numbers(numbers):
    if not numbers:
        return None, "No numbers provided"

    total = sum(numbers)
    average = total / len(numbers)
    return total, average, len(numbers)
```

# The __main__ Block

- The __main__ block allows code to run only when the script is executed directly, not when imported.

```
def add_numbers(a, b):
    return a + b

def subtract_numbers(a, b):
    return a - b

def main():
    # Main program logic
    print("Calculator Program")
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))

    sum_result = add_numbers(num1, num2)
    diff_result = subtract_numbers(num1, num2)

    print(f"Sum: {sum_result}")
    print(f"Difference: {diff_result}")

# This runs only when script is executed directly
if __name__ == "__main__":
    main()
```

# Exercise

- Create a program with the following functions:
  - **`get_student_info()`** - Gets student name and scores
  - **`calculate_average(scores)`** - Calculates average of scores
  - **`determine_grade(average)`** - Determines letter grade based on average
  - **`save_student_record(name, average, grade, filename)`** - Saves to file
  - **`main()`** - Orchestrates the program
- Expected Output:

  Enter student name: Alice
  Enter score 1: 85
  Enter score 2: 92
  Enter score 3: 78
  Alice's average: 85.0
  Grade: B
  Student record saved to grades.txt

# Exercise (Optional)

- Video Game High Score Tracker. Requirements:
  - Use proper function signatures with appropriate parameters
  - Implement multiple return values where appropriate
  - Follow the "one task per function" principle
  - Use the __main__ block structure
  - Save results to individual files for each player
  - Make it fun with emojis and exciting messages!
- Refer to code repository for details.