**Python Fundamentals**

**Contents**

[Variables and Data Types 2](#_Toc170910435)

[Lists 3](#_Toc170910436)

[Dictionaries 3](#_Toc170910437)

[Functions 3](#_Toc170910438)

[Control Structures 4](#_Toc170910439)

[Conditional Statements 4](#_Toc170910440)

[Loops 4](#_Toc170910441)

[Input and Output 5](#_Toc170910442)

[Error Handling 5](#_Toc170910443)

[Grocery Shopping Budget and List Manager: Script Example 5](#_Toc170910444)

[Exercises 10](#_Toc170910445)

[Reading list 10](#_Toc170910446)

[Creating your own script: Personal Budget Tracker 11](#_Toc170910447)

[*Step 1: Set Up the Environment 11*](#_Toc170910448)

[*Step 2: Define Variables 11*](#_Toc170910449)

[*Step 3: Create a Function to Set the Budget 11*](#_Toc170910450)

[*Step 4: Create a Function to Add an Expense 11*](#_Toc170910451)

[*Step 5: Create a Function to Display All Expenses 11*](#_Toc170910452)

[*Step 6: Create a Function to Calculate the Remaining Budget 11*](#_Toc170910453)

[*Step 7: Create the Main Program Loop 11*](#_Toc170910454)

[*Final Step: Test the Program 11*](#_Toc170910455)

# Variables and Data Types

Variables are used to store information to be referenced and manipulated in a program. Python supports various data types to represent different kinds of data. For more information on data types, [click here](https://www.w3schools.com/python/python_datatypes.asp).

# Example

budget = 100.0  # float

item\_name = "apple"  # str

quantity = 5  # int

is\_on\_sale = True  # bool

int: Represents integer numbers (e.g., 5, -3).

float: Represents floating-point numbers or decimals (e.g., 100.0, -7.5).

For more information on integers and floats, [click here](https://www.w3schools.com/python/python_numbers.asp).

str: Represents strings, or sequences of characters (e.g., "apple", 'Hello').

For more information on strings, [click here](https://www.w3schools.com/python/python_strings.asp).

bool: Represents boolean values (True or False).

For more information on booleans, [click here](https://www.w3schools.com/python/python_booleans.asp).

# Lists

Lists are ordered collections of items that are changeable and allow duplicate elements. Lists are defined using square brackets [].

# Example

shopping\_list = ["apple", "banana", "cherry"]

shopping\_list.append("date")  # Adds "date" to the list

print(shopping\_list)  # Output: ['apple', 'banana', 'cherry', 'date']

Use shopping\_list[index] to access elements by index, e.g. 0, 1, 2, 3.

Use methods like append(), remove(), and pop() to modify lists.

For more information on lists, [click here](https://www.w3schools.com/python/python_lists.asp).

# Dictionaries

Dictionaries are collections of key-value pairs. Each key is unique, and values can be of any data type. Dictionaries are defined using curly braces {}.

# Example

recipes = {

    "Pasta": ["pasta", "tomato sauce", "cheese"],

    "Salad": ["lettuce", "tomato", "cucumber", "dressing"]

}

print(recipes["Pasta"])  # Output: ['pasta', 'tomato sauce', 'cheese']

Access values using recipes[key].

Add or update values using recipes[key] = value.

For more information on dictionaries, [click here](https://www.w3schools.com/python/python_dictionaries.asp).

# Functions

Functions are blocks of reusable code that perform a specific task. Functions are defined using the def keyword.

# Example

def greet(name):

    return f"Hello, {name}!"

print(greet("Alice"))  # Output: Hello, Alice!

Functions can take parameters (inputs) and return a value.

Use def function\_name(parameters): to define a function.

For more information on functions, [click here](https://www.w3schools.com/python/python_functions.asp).

# Control Structures

Control Structures manage the flow of a program. Common control structures include conditional statements and loops.

## Conditional Statements

Conditional statements execute code based on whether a condition is True or False.

# Example

age = 20

if age >= 18:

    print("You are an adult.")

elif age > 12:

    print("You are a teenager.")

else:

    print("You are a child.")

if, elif, and else keywords are used to create conditional statements. For more information, [click here](https://www.w3schools.com/python/python_conditions.asp).

## Loops

Loops are used to execute a block of code repeatedly.

# For Loop Example

for i in range(5):

    print(i)  # Output: 0 1 2 3 4

# While Loop Example

count = 0

while count < 5:

    print(count)

    count += 1  # Output: 0 1 2 3 4

for loops iterate over a sequence (e.g., list, range). [Click here](https://www.w3schools.com/python/python_for_loops.asp) for more information.

while loops continue as long as a condition is True. [Click here](https://www.w3schools.com/python/python_while_loops.asp) for more information.

# Input and Output

Input allows the program to receive data from the user, and output displays data to the user.

# Example

name = input("Enter your name: ")  # Input from the user

print(f"Hello, {name}!")  # Output to the user

input(prompt) is used to get user input as a string.

print() is used to display output.

# Error Handling

Error Handling allows a program to handle errors gracefully using try and except.

# Example

try:

    result = 10 / 0

except ZeroDivisionError:

    print("Cannot divide by zero")

try block contains code that might raise an exception.

except block contains code that runs if an exception occurs.

# **Grocery Shopping Budget and List Manager: Script Example**

# Define variables

budget = 100.0

# A variable 'budget' is set to 100.0, representing the total money available for shopping.

shopping\_list = []

# List to store shopping items  
# 'shopping\_list' is an empty list that will hold the items you want to buy.

recipes = {

    "Pasta": ["pasta", "tomato sauce", "cheese"],

    "Salad": ["lettuce", "tomato", "cucumber", "dressing"],

    "Omelette": ["eggs", "cheese", "milk"]

}

# 'recipes' is a dictionary where each recipe name is a key, and the value is a list of ingredients.

# Welcome message

print("Welcome to the Grocery Shopping Budget Manager!")

# Prints a welcome message to the user.

# Function to display current shopping list and budget

def display\_status():

"""

This function displays the current shopping list and remaining budget.

It iterates over the shopping list, prints each item and its price, and calculates the remaining budget.

"""

    print("\nCurrent Shopping List:")

# Prints a heading for the shopping list.

    for item in shopping\_list:

        print(f"- {item['name']} (${item['price']})")

# Loops through each item in 'shopping\_list' and prints the name and price of each item.

    print(f"Remaining Budget: ${budget - sum(item['price'] for item in shopping\_list):.2f}")

# Calculates and prints the remaining budget by subtracting the total cost of items in 'shopping\_list' from 'budget'.

# Function to add an item to the shopping list

def add\_item(name, price):

"""

This function adds an item to the shopping list if the budget allows.

Parameters:

- name: The name of the item (str)

- price: The price of the item (float)

It checks if the price of the item is within the remaining budget and adds it to the shopping list if possible.

"""

    global budget

# This allows the function to modify the global 'budget' variable.

    if price <= (budget - sum(item['price'] for item in shopping\_list)):

        shopping\_list.append({"name": name, "price": price})

# If the price is within the remaining budget, it adds the item to 'shopping\_list'.

        print(f"Added {name} to the shopping list.")

# Prints a confirmation message.

    else:

        print(f"Cannot add {name}. Not enough budget.")

# If the price is not within the remaining budget, it prints an error message.

# Function to create a shopping list based on selected recipes

def create\_shopping\_list():

"""

This function creates a shopping list based on the recipes selected by the user.

It displays available recipes, asks the user to select recipes, and then adds the necessary ingredients to the shopping list.

"""

    print("\nAvailable Recipes:")

# Prints a heading for the list of available recipes.

    for recipe in recipes.keys():

        print(f"- {recipe}")

# Loops through each recipe in 'recipes' and prints the recipe name.

    selected\_recipes = input("Enter the names of the recipes you want to make (comma separated): ").split(",")

# Asks the user to input the names of the recipes they want to make, separated by commas.

# Splits the input string into a list of recipe names.

    ingredients\_needed = set()

# Creates an empty set to hold the unique ingredients needed.

    for recipe in selected\_recipes:

        recipe = recipe.strip()

# Removes any extra spaces from the recipe name.

        if recipe in recipes:

            ingredients\_needed.update(recipes[recipe])

# If the recipe is in 'recipes', add its ingredients to 'ingredients\_needed'.

        else:

            print(f"Recipe {recipe} not found.")

# If the recipe is not found, print an error message.

    for ingredient in ingredients\_needed:

        price = float(input(f"Enter the price for {ingredient}: "))

# Asks the user to input the price for each unique ingredient needed.

        add\_item(ingredient, price)

# Adds the ingredient and its price to the shopping list.

# Main program loop

while True:

"""

This is the main loop of the program.

It displays options to the user, gets the user's choice, and performs actions based on the choice.

The loop continues until the user chooses to exit.

"""

    print("\nOptions:")

    print("1. Display current shopping list and budget")

    print("2. Add item to shopping list")

    print("3. Create shopping list from recipes")

    print("4. Exit")

# Prints the menu options for the user to choose from.

    choice = input("Enter your choice (1-4): ")

# Asks the user to enter their choice of action.

    if choice == '1':

        display\_status()

# If the user enters '1', call 'display\_status()' to show the current shopping list and budget.

    elif choice == '2':

        item\_name = input("Enter the name of the item: ")

# Asks the user to input the name of the item they want to add.

        item\_price = float(input("Enter the price of the item: "))

# Asks the user to input the price of the item they want to add.

        add\_item(item\_name, item\_price)

# Calls 'add\_item()' to add the item and its price to the shopping list.

    elif choice == '3':

        create\_shopping\_list()

# If the user enters '3', call 'create\_shopping\_list()' to create a shopping list based on recipes.

    elif choice == '4':

        print("Thank you for using the Grocery Shopping Budget Manager!")

# If the user enters '4', print a thank you message and exit the loop.

        break

# Breaks out of the while loop to end the program.

    else:

        print("Invalid choice. Please try again.")

# If the user enters an invalid choice, print an error message and prompt again.

# **Exercises**

## Reading list

Data types: **integers** and **floats;** [click here](https://www.w3schools.com/python/python_numbers.asp) to access.

Data types: **strings;** [click here](https://www.w3schools.com/python/python_strings.asp) to access.

Data types: **booleans;** [click here](https://www.w3schools.com/python/python_booleans.asp) to access.

**Lists**; [click here](https://www.w3schools.com/python/python_lists.asp) to access.

**Dictionaries**; [click here](https://www.w3schools.com/python/python_dictionaries.asp) to access.

**Functions**; [click here](https://www.w3schools.com/python/python_functions.asp) to access.

**If statements**; [click here](https://www.w3schools.com/python/python_conditions.asp) to access.

**For loops**; [click here](https://www.w3schools.com/python/python_for_loops.asp) to access.

**While loops**; [click here](https://www.w3schools.com/python/python_while_loops.asp) to access.

## Creating your own script: Personal Budget Tracker

### Step 1: Set Up the Environment

Start by setting up your Python environment. Create a new Python file named budget\_tracker.py.

### Step 2: Define Variables

Define two variables: monthly\_budget and expenses. Initialize monthly\_budget to 0.0 and expenses as an empty list.

### Step 3: Create a Function to Set the Budget

Write a function called set\_budget() that allows the user to input their monthly budget. This function should update the monthly\_budget variable.

### Step 4: Create a Function to Add an Expense

Write a function called add\_expense() that allows the user to input the description and amount of an expense. This function should add a dictionary with the expense details to the expenses list.

### Step 5: Create a Function to Display All Expenses

Write a function called display\_expenses() that prints all the expenses in the expenses list.

### Step 6: Create a Function to Calculate the Remaining Budget

Write a function called remaining\_budget() that calculates the remaining budget and prints it.

### Step 7: Create the Main Program Loop

Write a while True loop to display menu options to the user. The loop should allow the user to set the budget, add an expense, display expenses, calculate the remaining budget, or exit the program. Use input() to get the user's choice and call the appropriate function based on the input.

### Final Step: Test the Program

Run the script and test each feature to ensure everything works as expected. Make sure you can set the budget, add expenses, display expenses, calculate the remaining budget, and exit the program.