Week 2: COMP-801 - Integrated Computing Practice

Agenda

- Feedback on AR1
- Command-line interface (CLI) basics
- Getting started on **lab1**
- Review and highlight concepts from AR1 and AR2

Feedback on AR1

- Browse through AR1 student progress report
- Answer and discuss select AR1 questions

Command-Line Interface (CLI) Basiscs

- Review CLI Basics resource in Week 2 module in Canvas
- Form teams of 2 students to collaborate
  - Team members should have the same operating systems (if possible)
- Open a CLI at the system level, switch to bash, try to deactiavte `conda`
- Install `pytest`

Getting `lab1` Assignment

In the CLI (terminal) of our operating system:

- Change directory to `home directory`
- Create `comp801` in home directory
- Create subfolders `labs`, `homework`, `practice` in `comp801`
- Follow Lab 1 instructions to get access tot he **`lab1`** starter project.

Visual Studio: Getting started

Open VS Code, from **File --> Open Folder...**, select **lab1**

- Clone starter project **lab1** as instructed in Lab 1 description (Canvas Lab1 page)
- Examine project structure: `core.py`, `client.py`, `test_div_nums.py` and `test_last_chars.py`
- Run each module
- Run the tests using `pytest` tool

Lab1: First Development Step

- Document `.py` modules as instructed in Lab 1
- Do version control for this step of development
- Write the 2nd testing function for `div_nums()` function

📖 Values and Variables

- **Value** - Unit of data, such as
  - a number (integer `3`, float `7.5`)
  - a string (`"hi"`, `'hi'`, `"""hi"""`)
  - a Boolean value (`True`, `False`)
  - a list of values (`[1, 2, 3]`, `['hi', True, 3.5]`)
  - and more
- **Variable**
  - Name (identifier) that references an object

📖 Operators

- **Operators**
  - arithmetic (`2 + 3`)
  - comparison (`2 < 3` , `2 == 3`)
  - Boolean (logical) (`not True` , `True and False`)
  - sequence operations: indexing `[]` , slicing `[ : ]` , concatenation `+`
  - membership operation: `in`
  - and more ...

📖 Expressions and Statements

- **Expression**
  - Computation that **produces a SINGLE value**
  - Made up of values, variables, operators, and other expressions
- **Statement**
  - Computation that **performs an action**
  - Made up of keywords, delimiters, expressions, and other statements

📖 Flow of Control

Control flow of execution in a program is **sequential** *UNLESS* it is altered by:

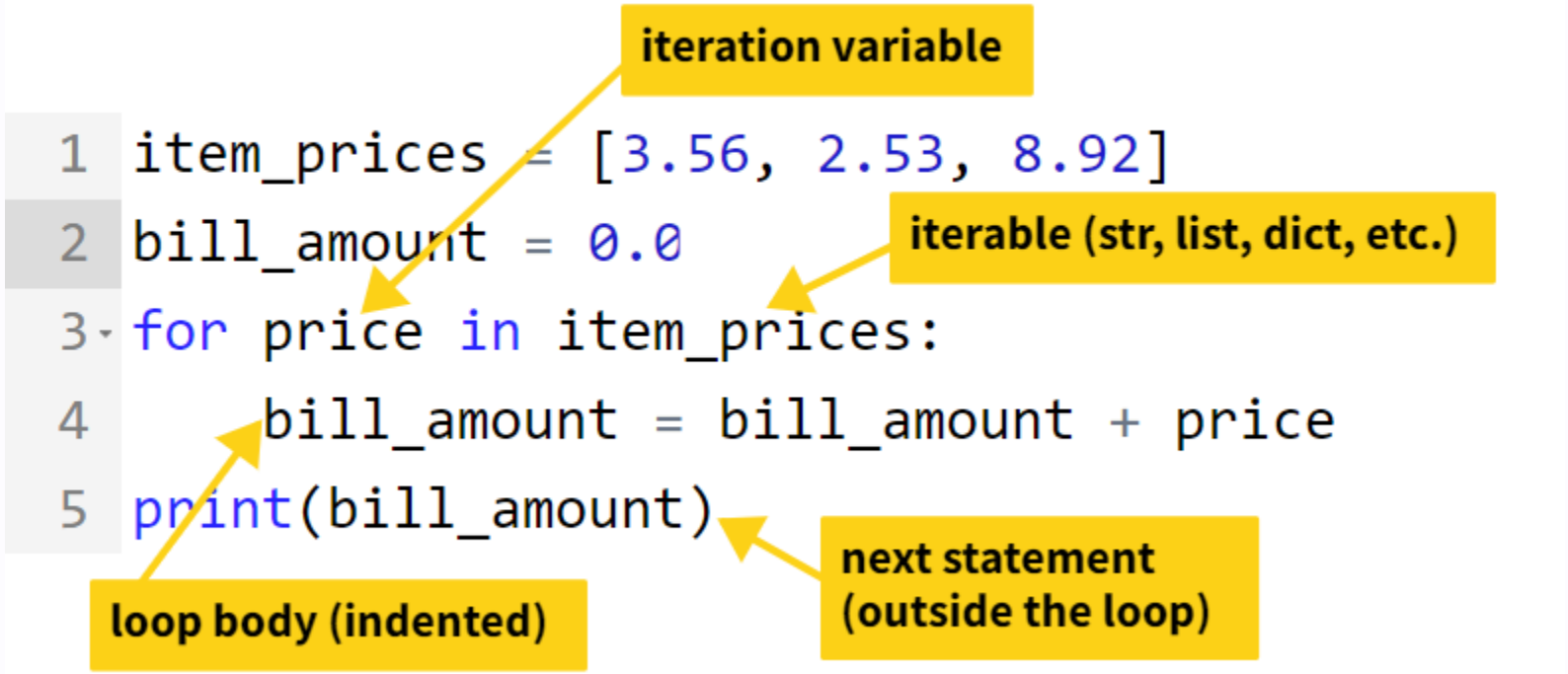
- **Function calls**
- **Loops**
- **Conditionals**
- and other statements (e.g. `break`, `continue`, `return`, `except`, `raise`)

📖 Control Structures

Alter the flow of execution control.

- **Loop**
  - repeats statements in the body of the loop until termination condition is reached
- **Conditional**
  - selects a branch of statements to execute based on selection condition

📖 For loop Components



📖 Functions

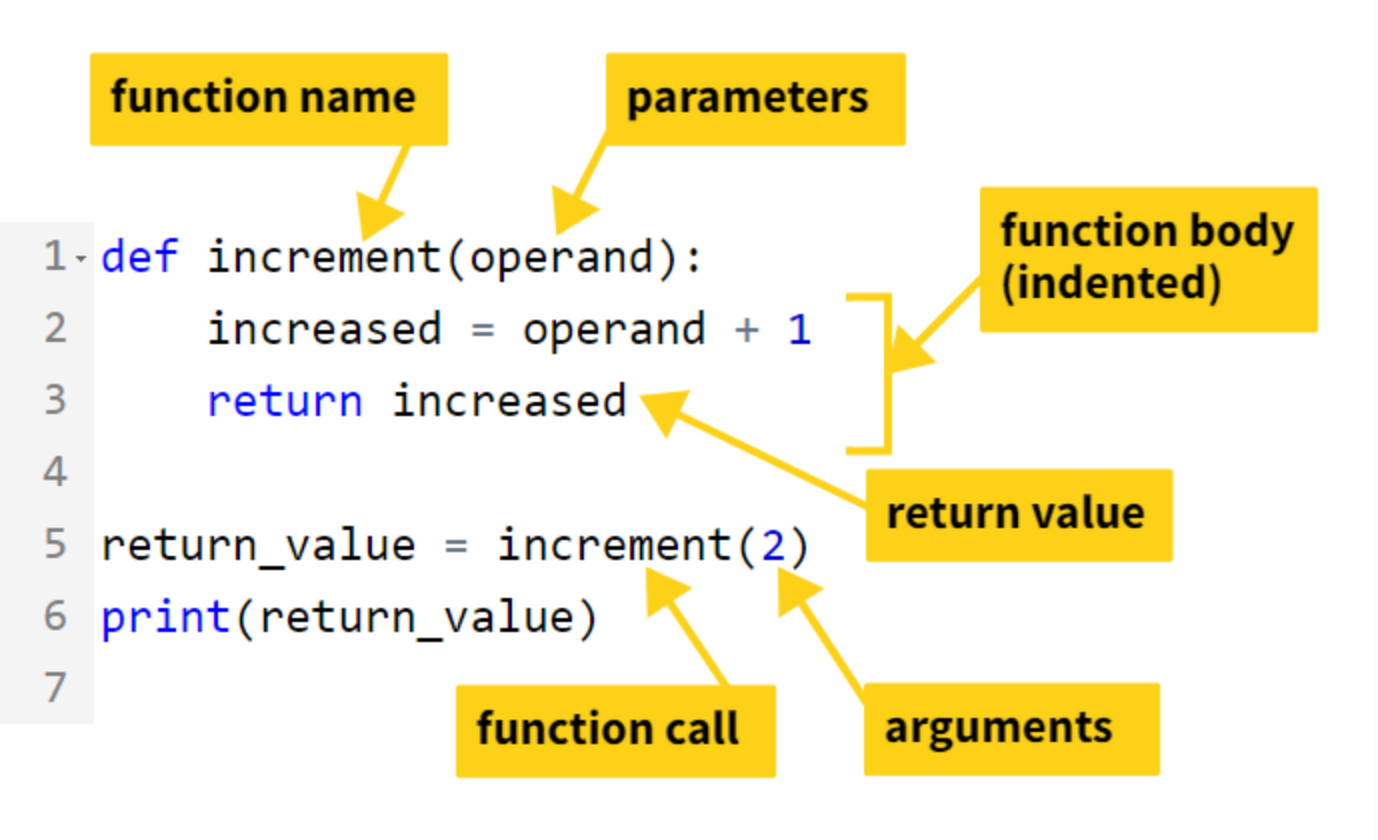
A **function**

- is a named sequence of statements that perform a useful task
- may be defined with **parameters**
- may OR may not produce a **return value**

📖 Function Definition

- **Function definition** might have parameters
- **Parameters** are variables or names
  - defined in the **function header**
  - refer to the values passed as **arguments** by the **call**
  - used in the **function body**

📖 Function Definition Example



📖 Function Call

Alters the control flow

- jumps to the **function definition**
- executes function's statements, and
- resumes from statement following function call

📖 Function Call Arguments

A function call might require arguments

Argument is the value passed into a function when it is called

- as simple as a number or string or other literal
- OR an expression
- OR another function call that returns a value

📖 Function Return Value

- A **return value** is the value produced when a function is executed
- Not all functions may return a value

Document Your Code

- Write a **docstring** (triple double quotes)
  - Module: at the top of the module file
  - Function definition: **indented** and **below** the function header
  - Class definition: **indented** and **below** the class header
  - Method definition: **indented** and **below** the method header

What to Document

- In the function documentation docstring, explain concisely
  - **what** the function does, with no details about **how** it does it
  - Parameters:
    - brief description and data type
  - Returns:
    - what the function returns (if any) and its data type

docstring Example

```
def size(sentence):
    """
    Return the size of sentence.

    :param sentence: str
    :return: int, size of the sentence
    """
    ...
```

PEP 8 Naming Conventions

- PEP 8 Style Guide for Python Code
- Google Python Style Guide

Type	Public
Packages	lower_with_under
Modules	lower_with_under
Classes	CapWords
Exceptions	CapWords
Functions	lower_with_under()
Global/Class Constants	CAPS_WITH_UNDER
Global/Class Variables	lower_with_under
Instance Variables	lower_with_under
Method Names	lower_with_under()
Function/Method Parameters	lower_with_under
Local Variables	lower_with_under

Syntax and Style Code Analyzers

- **pylint**
  - Checks for errors, coding standards, "code smells"
- **pycodestyle**
  - Checks some style conventions in PEP 8
  - Does NOT check naming conventions and docstrings
- **flake8**
  - Checks all of the above with plugins
- Install VS Code extensions: `black formatter`, `pylance`, `pylint`, and `flake8`
- Using `pip` (or `pip3`), installe `pycodestyle`

Code Smells

- CodeSmell, Martin Fowler and Ken Beck
- Finding Code Smells, Al Swaigart