Lab 2: Programming in Python

Learning objectives

After completing this lab you will be able to:

- Understand Python's syntax, data types, and basic programming concepts like variables and basic operations.
- Use control structures like loops (`for` and `while`), `if` statements, and `else` clauses to control the flow of your programs.
- Understand the concept of functions, how to define them, pass arguments, and return values.

The purpose of this laboratory exercise is to provide students with a fundamental understanding of Python programming. These objectives aim to establish a strong foundation in Python programming concepts and skills, including flow control structures and functions. To achieve this, ESP32 microcontroller and MicroPython version are being used in the lab.

Table of contents

Components list

- ESP32 board, USB cable

Pre-Lab preparation

1. If you don't have any, create a free account on GitHub.

2. For future synchronization of local folders with GitHub, download and install git. Git is free, open source, and available on Windows, Mac, and Linux platforms. Window users may also need to use the Git Bash application (installed automatically with git) for command line operations.

Part 1: GitHub

GitHub serves as a platform for hosting code, facilitating collaboration, and managing version control. It enables you and your collaborators to work together on projects, retain a history of all prior changes, create distinct branches, and offers a multitude of additional features.

1. In GitHub, create a new public repository titled **esp-micropython**. Initialize a README, Python template `.gitignore`, and MIT license.

2. Use any available Git manuals, such as Markdown Guide, Basic Syntax and add the following sections to your README file.

    - Headers H1, H2, H3
    - Emphasis (*italics*, **bold**)
    - Lists (ordered, unordered)
    - Links
    - Table
    - Listing of Python source code (with syntax highlighting)

3. Use your favorite file manager and run the Git Bash (Windows) or Terminal (Linux) application inside your home folder `Documents`.

4. With help of Git command, clone a local copy of your public repository.

   > **Important:** To avoid future problems, never use national characters (such as éščřèêö, ...) and spaces in folder- and file-names.
   >
   > **Help:** Useful git command is `git clone` - Create a local copy of remote repository. This command is executed just once; later synchronization between remote and local repositories is performed differently.
   >
   > Useful bash commands are `cd` - Change working directory. `mkdir` - Create directory. `ls` - List information about files in the current directory. `ls -a` - List information aout all files in the current directory. `pwd` - Print the name of the current working directory.

   ```
   ## Windows Git Bash or Linux:
   git clone https://github.com/your-github-account/esp-micropython
   cd esp-micropython/
   ls -a
   ## You should see these three files
   .gitignore  LICENSE  README.md
   ```

5. Set username and email for your repository (values will be associated with your later commits):

   ```
   git config user.name "your-git-user-name"
   git config user.email "your-email@address.com"
   ```

   You can verify that the changes were made correctly by:

   ```
   git config --list
   ```

6. (Optional) Using branches in Git is a fundamental concept that allows you to work on different features or aspects of a project simultaneously without affecting the main codebase. Using branches in Git allows for efficient code management, parallel development, and collaboration in a team setting. It helps prevent conflicts between different features or bug fixes being worked on simultaneously. Here is the basic way to use branches in Git.

a. **Create a New Branch:** To create a new branch, use the following command:

```
git branch branch_name
```

Replace `branch_name` with the name you want to give to your new branch. This command creates a new branch but doesn't switch to it yet.

b. **Switch to a Branch:** To switch to a branch, use the following command:

```
git checkout branch_name
```

Replace `branch_name` with the name of the branch you want to switch to. After executing this command, you're working in the context of the chosen branch.

Alternatively, you can use a single command to create and switch to a new branch:

```
git checkout -b new_branch_name
```

c. **Make Changes:** Now that you're on the new branch, you can make changes to your project. These changes are isolated to this branch and won't affect the main or other branches.

d. **Commit Changes:** After making changes, commit them to your branch using the following commands:

```
git add .
git commit -m "Descriptive commit message"
```

The `git add .` command stages your changes, and `git commit` records them with a meaningful commit message.

e. **Push Branch (Optional):** If you want to share your branch and collaborate with others, you can push it to a remote repository:

```
git push origin branch_name
```

f. **Merge or Rebase (Optional):** Once your changes on the branch are complete, you can merge the branch back into the main branch or another target branch using commands like `git merge` or `git rebase`.

g. **Delete Branch (Optional):** After the branch's changes have been merged or are no longer needed, you can delete it:

```
git branch -d branch_name
```

The `-d` flag stands for "delete." If the branch contains unmerged changes, you may need to use `-D` instead: `git branch -D branch_name`.

Part 2: Basic operations in Python

1. Use micro USB cable and connect the ESP32 board to your computer. Run Thonny IDE and check if selected interpreter is Micropython (ESP32). If not, go to menu **Run > Select interpreter... > Interpreter** and select `ESP32` or `ESP8266`. Click on red **Stop/Restart** button or press the on-board reset button if necesary.

2. In the **Shell** window, attempt the following arithmetic, binary, and string operations using variables. Note that you can use the `print()` function to display values or text.

```
# Arithmetic operations
>>> 10/3
3.333333
>>> 10//3
3
>>> 10%3
1
>>> 10*3
30
>>> 10**3
1000
```

```
# Binary operations
>>> a = 5  # Binary: 0101
>>> b = 3  # Binary: 0011
>>> result = a & b
>>> print(result)

>>> result = a | b
>>> print(f"Bitwise OR result is {result}")

>>> result = a ^ b
>>> print(f"Bitwise XOR result is {result}")

>>> result = ~a
>>> print(f"Bitwise NOT of {a} is {result}")

>>> left_shifted = a << 1
>>> print(f"Left shift {a} is {left_shifted}")
```

```
# String operations
>>> str1 = "Hello, "
>>> str2 = "world!"
>>> text = str1 + str2
>>> print(text)

>>> result = str1 * 4
>>> print(result)

>>> length = len(result)
>>> print(length)

>>> first_char = text[0]
>>> third_char = text[2]
>>> last_char = text[-1]
>>> print(...)
```

Part 3: Functions in Python

1. In Thonny IDE, create a new source file in menu **File > New Ctrl+N**, save it as `functions.py` to your local folder. Program the function to display an `arrow` of symbols and run the application by **Run > Run current script F5**. Note that you can use **if** statements and **for** loops.

```
def print_arrow(width, symbol):
    """
    Print an arrow made of symbols with a defined width.

    Args:
        width (int): The width of the arrow.
        symbol (str): The symbol used to create the arrow.

    Example:
        print_arrow(5, "*") would print:
        *
        **
        ***
        ****
        *****
        ****
        ***
        **
        *
    """
    # Complete the code

# Example usage
arrow_width = 5
print_arrow(arrow_width, "*")
```

2. Create a function `my_factorial(n)` to calculate a factorial of input `n`.

3. Create a function `def my_factorial(n)` to solve the quadratic equation.

```
def solve_quadratic_eq(a, b, c):
    """
    Solve a quadratic equation of the form ax^2 + bx + c = 0.

    Args:
        a (float): Coefficient of x^2.
        b (float): Coefficient of x.
        c (float): Constant term.

    Returns:
        tuple: A tuple containing the real or complex roots.

    Example:
        solve_quadratic_eq(1, 5, 1) returns the roots (approximately):
        (-0.2087, -4.7912)
    """
    # Complete the code

# Example usage
a = 1
b = 5
c = 1
roots = solve_quadratic_eq(a, b, c)
print(f"Roots: {roots}")
```

If you require a library function in Python, you need to import the module that contains it.

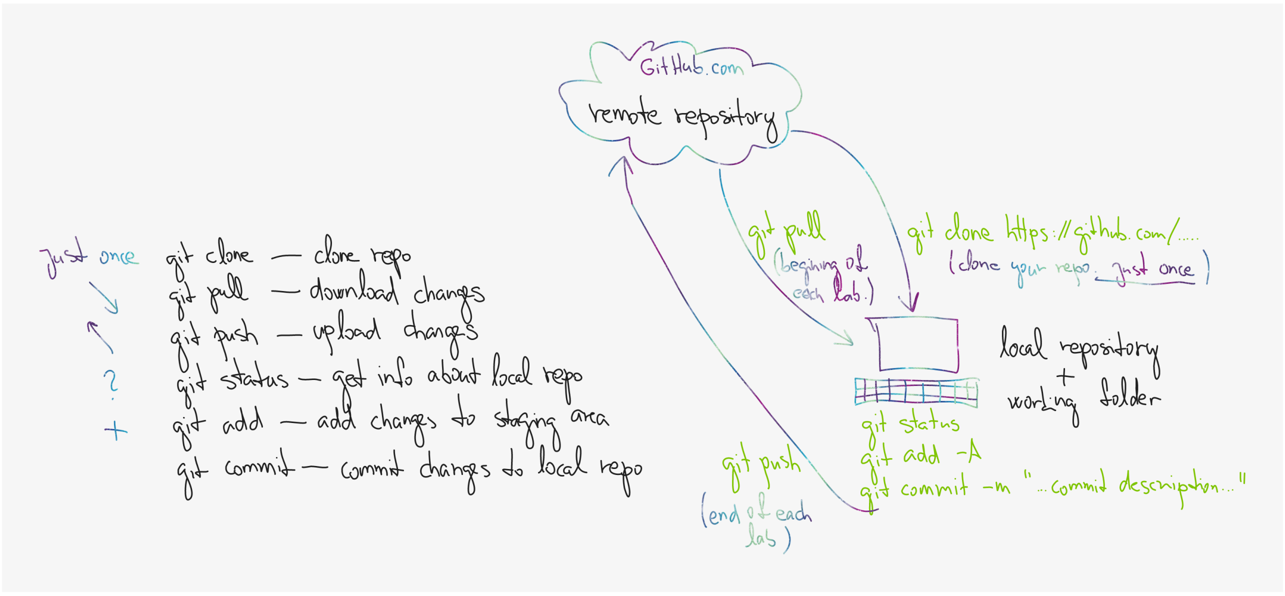
```
import math   # Import mathematical module
import cmath  # Complex numbers' math

math.sqrt()   # Call the function
```

4. After completing your work, ensure that you synchronize the contents of your working folder with both the local and remote repository versions. This practice guarantees that none of your changes are lost. You can achieve this by using Git commands to add, commit, and push all local changes to your remote repository. Check GitHub web page for changes.

> **Help:** Useful git commands are `git status` - Get state of working directory and staging area. `git add` - Add new and modified files to the staging area. `git commit` - Record changes to the local repository. `git push` - Push changes to remote repository. `git pull` - Update local repository and working folder. Note that, a brief description of useful git commands can be found here and detailed description of all commands is here.

```
## Windows Git Bash or Linux:
$ git status
$ git add -A
$ git status
$ git commit -m "Creating functions in Python"
$ git status
$ git push
$ git status
```


GitHub.com
remote repository
Just once
git clone — clone repo
git pull — download changes
git push — upload changes
git status — get info about local repo
git add — add changes to staging area
git commit — commit changes to local repo
git pull
(begining of each lab.)
git clone https://github.com/.....
(clone your repo. Just once)
local repository
+
working folder
git status
git add -A
git commit -m "...commit description..."
git push
(end of each lab)

(Optional) Experiments on your own

1. Write a Python function that determines whether a given number is prime or not and generate all prime numbers up to 1000.
2. Implement a function to generate Fibonacci numbers. This is a classic sequence where each number is the sum of the two preceding ones (0, 1, 1, 2, 3, 5, 8, ...).

References

1. Markdown Guide, Basic Syntax
2. learnpython.org
3. Tomas Fryza. Useful Git commands
4. Joshua Hibbert. Git Commands