Programming Conventions

CS 181
Object-oriented programming

Alex Petrovici

Content

Operating System

We'll use Unix based systems:

- Linux / Mac OS
- Windows -> How to install Linux on Windows with WSL
  - Install wsl (if you dont have it already)

```
wsl --install
```

  - Install Ubuntu

```
wsl.exe --install ubuntu
```

Shell / Introduction

Why zsh ?

- zsh is much more configurable
- zsh is also used on Mac

How to change from bash to zsh ?

What shell do I currently have ?

```
echo $SHELL
```

Expectation:

```
/usr/bin/bash
```

Shell / How to install zsh

If you have bash then install zsh:

1. Update the package source list and updates all the packages presently installed, with this command.

```
sudo apt update && sudo apt upgrade -y
```

-y will say yes to all the petitions.

2. Install zsh

```
sudo apt-get install zsh -y
```

3. Check that zsh has installed correctly:

```
zsh --version
```

```
Expectation: zsh 5.9 (x86_64-ubuntu-linux-gnu)
```

Shell / How to install zsh

4. Set Zsh as default shell

```
chsh -s $(which zsh)
```

5. Reboot or log off.

Check that zsh has installed correctly:

```
echo $SHELL
```

Expectation

```
/usr/bin/zsh
```

Shell: How to install zsh: If you want to go back to bash

In case you need to uninstall.

1. Set the shell to bash

```
chsh -s /bin/bash
```

2. Then uninstall zsh.

```
sudo apt remove zsh
```

Shell / Recomendation: using ohmyzsh

Chek the updated installations instructions at: `https://ohmyz.sh/`

Install ohmyzsh

```
sh -c "$(curl -fsSL
↪  https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
```

Now you'l be able to do:

- Move across folders with **. or .. or ...** depending on the amount of dirs that you want to go upwards.
- Autocomplete directories, filenames and commads.
- Use **z** (autojump) to quickly cd into a frequently used folder just by typing part of its name.

Shell / Recomendation: using powerlevel10k

Installation instructions:
https://github.com/romkatv/powerlevel10k?tab=readme-ov-file#installation

1. Clone the repository

```
git clone --depth=1 https://github.com/romkatv/powerlevel10k.git
↪  ${ZSH_CUSTOM:-$HOME/.oh-my-zsh/custom}/themes/powerlevel10k
```

Shell / Recomendation: using powerlevel10k

2. Update .zshrc

Open the .zshrc file.

```
code ~/,zshrc
```

Then set the theme to be powerlevel10k:

```
export ZSH="$HOME/.oh-my-zsh"
ZSH_THEME="powerlevel10k/powerlevel10k"
```

Shell / Recomendation: using powerlevel10k

3. Install the fonts

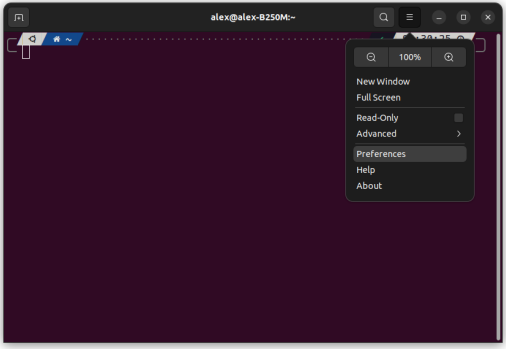
Instructions about installing the fonts: `https://github.com/romkatv/powerlevel10k?tab=readme-ov-file#fonts` Then download and install these fonts.

- MesloLGS NF Regular.ttf
- MesloLGS NF Bold.ttf
- MesloLGS NF Italic.ttf
- MesloLGS NF Bold Italic.ttf

Shell / Recomendation: using powerlevel10k

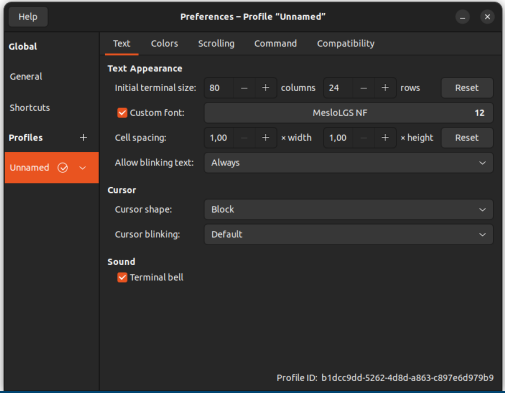
Configure the Terminal to use these fonts.

Click on Preferences.



Shell / Recomendation: using powerlevel10k

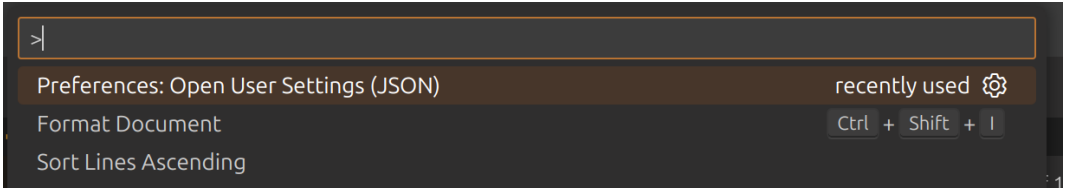
Use MesloLGS NF.



Shell / Recomendation: using powerlevel10k

Also configure the font of the terminal within Vscode.

Open **Preferences: Open User Settings (JSON)**



And add the line

```
"terminal.integrated.fontFamily": "MesloLGS NF",
```

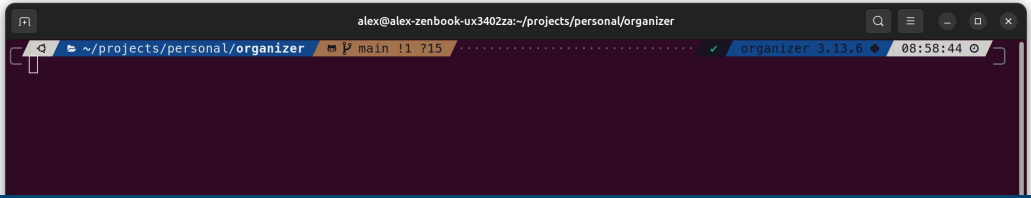
Shell / Recomendation: using powerlevel10k

Now you can configure p10k

Run in the terminal

```
p10k configure
```

When you're done you should be able to see at a glance the current location, git branch, how many files are modified or pending to commit, in which virtual environment we are located and its version.



Python environment / Introduction

- **Pipx**: Allows to install and run Python applications that are used as command-line tools, globally, in order to manage projects from the outside.
- **Pyenv**: Install different Python versions globally and locally to be used inside projects.
- **Pipenv**: Install module packages and creates environments for your projects. Recommended packaging tool by the official Python Packaging Authority

Source: `https://jacobsgill.es/python-package-management`

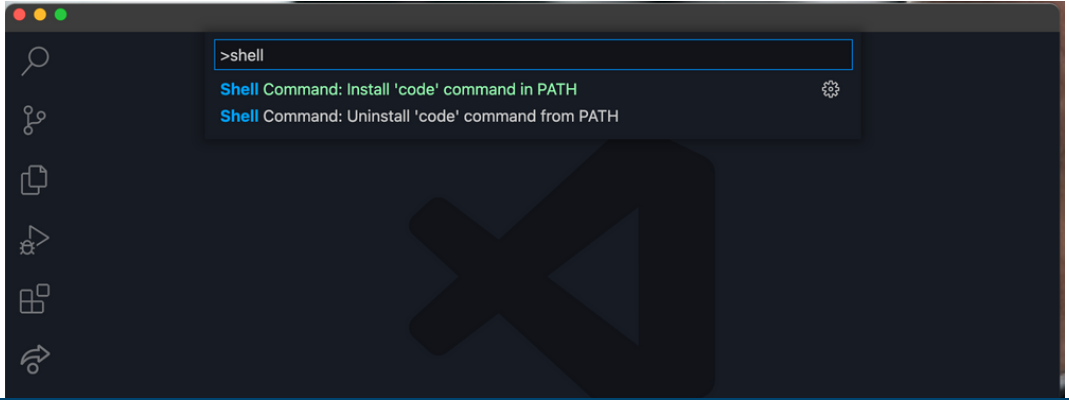
Python environment / Introduction

Make sure that we have the following dependencies:

```
sudo apt install -y \
build-essential \
libssl-dev \
zlib1g-dev \
libbz2-dev \
libreadline-dev \
libsqlite3-dev \
wget \
llvm \
libncurses5-dev \
libncursesw5-dev \
xz-utils \
tk-dev \
libffi-dev \
liblzma-dev
```

Python environment / Introduction

In order to edit files easier, add the vscode to path.



Python environment / pyenv

We'll manage the virtual environment with pyenv

```
curl https://pyenv.run | zsh
```

- `curl https://pyenv.run`: Uses curl to fetch data from the internet. In this case, it downloads the script from https://pyenv.run, which is a redirection to the installation script for pyenv.
- `| zsh`: This pipe (|) passes the output of the curl command (the script content) directly into zsh, then executes the script.

Python environment / pyenv

Add pyenv to shell startup script

Open the zshell configuration file:

```
code ~/.zshrc
```

And at the end the following lines:

```
export PYENV_ROOT="$HOME/.pyenv"
[[ -d $PYENV_ROOT/bin ]] && export PATH="$PYENV_ROOT/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"
```

Python environment / pyenv

After setting up pyenv, try installing the Python version:

```
pyenv install <version>
```

Usage:

```
pyenv install 3.13.5
```

Now install a global version of Python that is going to be used system-wide.

```
pyenv global 3.13.5
```

Python environment / pipx

For Ubuntu 23.04 or above:

```
sudo apt update
sudo apt install pipx -y
pipx ensurepath
```

Python environment / pipx

Then add pipx to the shell

Run:

```
pipx completions
```

Output:

```
Add the appropriate command to your shell's config file
so that it is run on startup. You will likely have to restart
or re-login for the autocompletion to start working.

bash:
    eval "$(register-python-argcomplete pipx)"

zsh:
    To activate completions for zsh you need to have
    bashcompinit enabled in zsh:

    autoload -U bashcompinit
    bashcompinit

    Afterwards you can enable completion for pipx:

    eval "$(register-python-argcomplete pipx)"
```

Python environment / poetry

Since we have installed pipx, the installation of Poetry is very simple:

```
pipx install poetry
```

After installing Poetry, we need to configure it to be able to read the current version of Python from PyEnv.

```
poetry config virtualenvs.create false
```

This makes Poetry install dependencies directly into whatever environment is already active.

```
poetry self add poetry-dotenv-plugin
```

It makes Poetry automatically load environment variables from a .env file whenever you run Poetry commands.

Python environment / poetry

In order to manage a project:

```
poetry init
```

And it will generate a **pyproject.toml** file that has the configuration of this environment.

Python environment / ruff (code formatting)

Many teams use the Ruff, Black or Pyink auto-formatter to avoid arguing over formatting. We will use Ruff (a python application that can be used globally or within a project).
Steps:

1. Install the VSCode extension

```
Name: Ruff
Id: charliermarsh.ruff
Description: A Visual Studio Code extension with support for the Ruff linter.
Version: 2024.20.0
Publisher: Astral Software
VS Marketplace Link: https://marketplace.visualstudio.com/items?itemName=charliermarsh.ruff
```

Python environment / ruff (code formatting)

2. Configure Ruff in VSCode to always autoformat.

```
"[python]": {
"editor.defaultFormatter": "charliermarsh.ruff",
"editor.formatOnSave": true,
"editor.formatOnType": true,
"editor.formatOnPaste": true,
},
"ruff.lint.run": "onSave",
"ruff.organizeImports": true,
"editor.codeActionsOnSave": {
"source.organizeImports": "explicit"
},
```

Python environment / ruff (code formatting)

3. How to add ruff to a new project

When we will create a new project, we'll install ruff dependency. Since we'll use poetry, it will be:

```
poetry add ruff --group dev
```

because we want to add it to the develop dependencies.

Python environment / docstrings

- Annotate code with type hints according to [PEP-484](https://peps.python.org/pep-0484/), and type-check the code at build time with a type checking tool like pytype or mypy.
- Google style docstring can be colorized with

```
Name: Python Docstring Highlighter
Id: rodolphebarbanneau.python-docstring-highlighter
Description: Syntax highlighting for python docstring.
Version: 0.2.3
Publisher: Rodolphe Barbanneau
VS Marketplace Link:
↪  https://marketplace.visualstudio.com/items?itemName=rodolphebarbanneau.python-docstring-hig
```

- Documentation:
  - https://google.github.io/styleguide/pyguide.html#383-functions-and-methods
  - https://peps.python.org/pep-0257/

Python environment / docstrings

```python
def fetch_smalltable_rows(
    table_handle: smalltable.Table,
    keys: Sequence[bytes | str],
    require_all_keys: bool = False,
) -> Mapping[bytes, tuple[str, ...]]:
    """Fetches rows from a Smalltable.

    Retrieves rows pertaining to the given keys from the Table instance
    represented by table_handle.  String keys will be UTF-8 encoded.

    Args:
        table_handle: An open smalltable.Table instance.
        keys: A sequence of strings representing the key of each table
        row to fetch.  String keys will be UTF-8 encoded.
        require_all_keys: If True only rows with values set for all keys will be
        returned.

    Returns:
        A dict mapping keys to the corresponding table row data
        fetched. Each row is represented as a tuple of strings. For
        example:
```

Git / Log in github via ssh

Generate a SSH key

Since `Support for password authentication was removed on August 13, 2021` we need to connect git to our github account via SSH access. Steps:
In the terminal:

```
ssh-keygen -t rsa -C your_email@example.com
```

- `ssh-keygen`: This is the program used to create the SSH keys.
- `-t rsa`: This option specifies the type of key to generate.

Git / Log in github via ssh

Hit Enter to save the key in the default location

```
~/.ssh/id_rsa
```

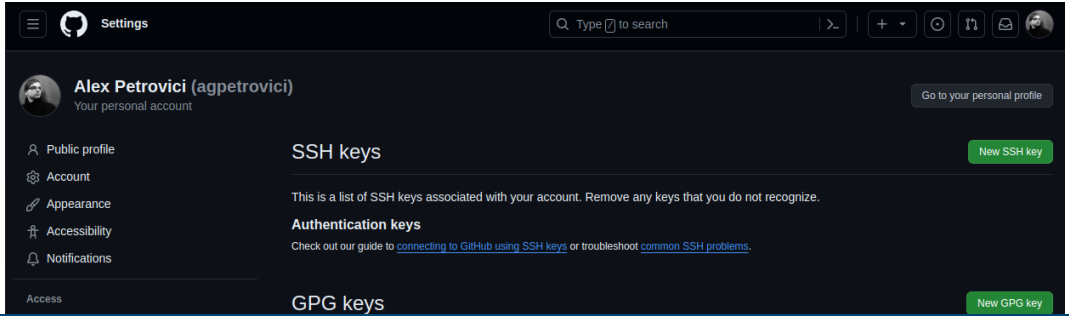
Hit Enter twice for no/empty passphrase.
Your SSH keys will be saved at the default location. Run the following command to view the generated public SSH key (will be pasted in github).

```
cat ~/.ssh/id_rsa.pub
```

Git / Log in github via ssh

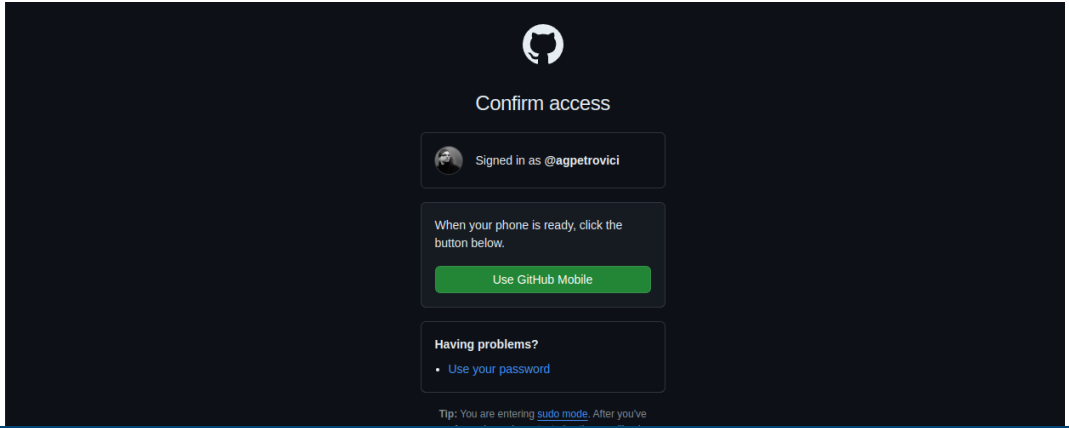
Add the SSH key to GitHub

- 1. Go to SSH Keys: `https://github.com/settings/keys`
- 2. Click on ‘New SSH key‘



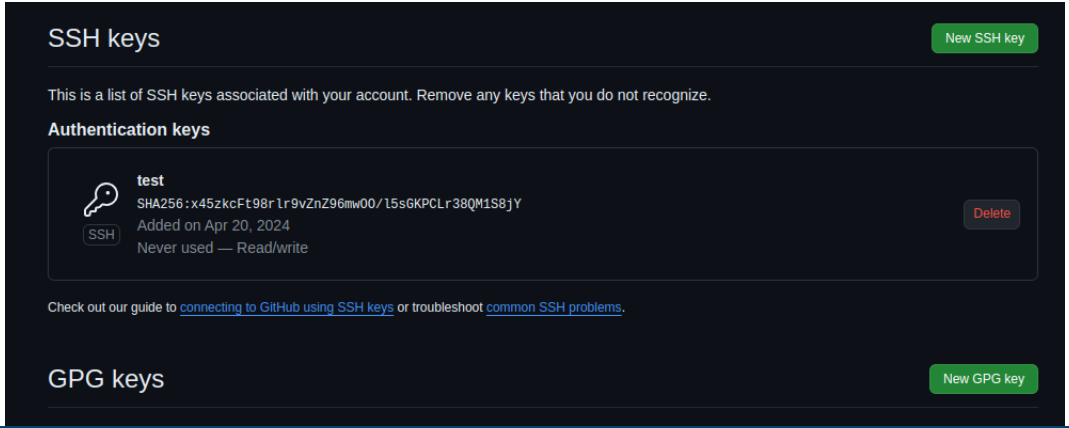
Git / Log in github via ssh

Confirm access:



Git / Log in github via ssh

Result:



Git / Log in github via ssh

Add github to the list of known_hosts

```
ssh-keyscan github.com >> ~/.ssh/known_hosts
```

- `github.com`: This specifies the host from which ssh-keyscan should collect the public SSH keys. You can also list multiple hosts or use IP addresses instead of domain names.
- »: is used to append the output to a file. If the file does not exist, it will be created. If you use a single ‘>‘ instead, it would overwrite the file each time, rather than appending to it.

It will copy the content from "GitHub’s SSH key fingerprints" `https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/githubs-ssh-key-fingerprints` and append them to `/.ssh/known_hosts`

Programming Principles /

*Our principles are the springs of our actions; our actions, the springs of our happiness or misery. Too much care, therefore, cannot be taken in forming our principles. – **Philip Skelton***

- **YAGNI** - You Aren't Gonna Need It
- **DRY** - Don't Repeat Yourself
- **KISS** - Keep It Simple, Stupid
- **SOLID**
- **LoD** - Law of Demeter
- **SLAP** - Single Level of Abstraction Principle
- **PIT** - Prefer Isolated Tests

Programming Principles / YAGNI - You Aren't Gonna Need It

- Don't add a functionality until it is necessary.
- Apply it during the feature planning and development stages.
- **Resist the urge to add features or functionality** "just in case" you might need it in the future.

Programming Principles / DRY - Don’t Repeat Yourself

- **DRY** code means that you don’t replicate a code and instead of that try using Abstraction to summarize the regular things in a single area.
- The DRY principle emphasizes that each piece of knowledge or logic must have a **single, unambiguous representation in the system**. This principle promotes maintainability and helps reduce errors.
- **DRY** can be used in almost all cases. If you see repetitive code or logic in your system, it might be a sign that you need to abstract that logic into functions, classes, or modules.

Programming Principles / KISS - Keep It Simple, Stupid

- The KISS principle suggests that the best solutions are often the simplest ones, and developers should strive to avoid unnecessary complexity.
- **When to use**: The KISS principle is best used when designing solutions and algorithms. Avoid over-engineering and always opt for the most straightforward solution that fulfills the requirement.

Programming Principles / SOLID

Single-Responsibility Principle

- A class should have only one reason to change.

Open-Closed Principle

- Classes should be open for extension but closed for modification.

Liskov Substitution Principle

- Subtypes should be substitutable for their base types without altering the correctness of the program.

Interface Segregation Principle

- Clients should not be forced to depend on interfaces they don't use.

Dependency Inversion Principle

- High-level modules should not depend on low-level modules; both should depend on abstractions.

- **SOLID** principles are essential when designing **object-oriented systems**, and they also apply to other programming paradigms. They should be used whenever you're defining classes and interfaces, planning software architecture, or refactoring existing code.

Programming Principles / LoD - Law of Demeter (Principle of Least Knowledge)

- The LoD states that an object should only communicate with its immediate neighbors and should have limited knowledge about other objects in the system.
- **When to use**: Apply the LoD when you are designing interactions between classes or objects in your system. It is particularly beneficial in complex systems where tight coupling can lead to increased dependencies and decreased modularity.

Programming Principles / PIT - Prefer Isolated Tests

- The PIT principle suggests that tests should be isolated and not dependent on one another. This principle leads to more reliable test suites and helps prevent cascading failures.
- **When to use**: Apply PIT when writing unit tests and integration tests. Make sure each test can run independently and in any order.

Programming Principles / SLAP - Single Level of Abstraction Principle

- The SLAP suggests that all statements in a function should be at the same level of abstraction, which improves readability and maintainability.
- **When to use**: Apply SLAP when writing functions or methods. If a function mixes high-level logic with low-level details, consider refactoring to separate these different levels of abstraction.