Advanced Software Engineering (LAB)

Stefano Forti

name.surname@di.unipi.it

Department of Computer Science, University of Pisa

Welcome to the ASE Lab!

Labs Plan (very tentative)

- Lab 1 – Python and GitHub 101
- Lab 2 – Flask Microservices — Homework 1 [individual*]
- Lab 3 – Containers and Docker 101
- Lab 4 – Testing
- Lab 5– From user stories to monolith — Homework 2 [groups*]
- Lab 6 – BPM with Camunda
- Lab 7 – Vulnerabilities with Bandit
- Lab 8 – … — Homework 3 [groups*]
- Lab 9 – …
- Lab 10 – … — Homework 4 [individual*]

*Plagiarism will be awarded an F grade ☹

Homework (my 2 cents)

Submissions

- Read the homework instructions carefully, and follow them. Answer to all requests.
- Submit **before** the deadline (Check you actually submitted as **draft or late submissions will not be graded**).
- Submit **all** required files and (please) follow the suggested format for naming them.

Reports

- There will be 200-300 word PDF reports* to submit along with assignment solutions.
- The report is part of the homework evaluation. Craft it carefully. Drafty reports will make you lose marks.
- Too long reports will make you lose marks (+10% tolerance on the number of words).
- (Please) Check English language with Grammarly or LanguageTools.

**For group assignments there will be individual Learning Journals and a single Group Report to submit.*

Lab goals

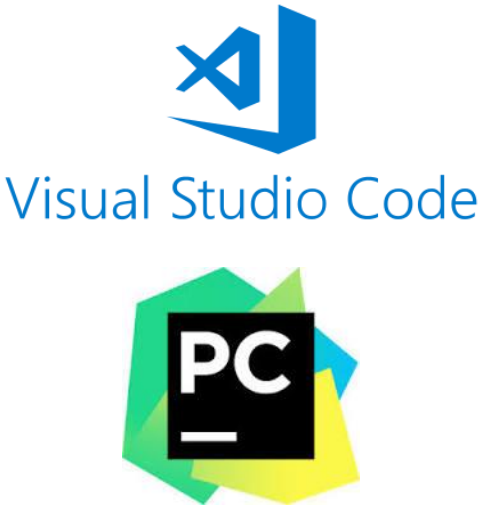
- Experiment state-of-the-art technologies in modern SWEng (practicing what you see in class).
- Work in groups to design and implement a prototype of a “real” microservice-based application.
- Make you more skilled, competent and experienced software engineers.

ABOVE ALL: Enjoy the learning process and keep things cool ☺

Checklist

Lab activities have been tested/designed under Ubuntu.
Try to install Ubuntu in a VM or natively.
If you use a different OS, you do «at your own risk»
(you will have to find your own workarounds sometimes…)

- Install **Python 3.9** [https://www.python.org/]
- Install an **IDE/editor** of your choice
  (e.g., Visual Studio Code [https://code.visualstudio.com/], PyCharm [https://www.jetbrains.com/pycharm/download/])
- Install **Git** (`sudo apt-get install git` or [https://git-scm.com/downloads])
- Create your **GitHub** account [http://github.com]



What will I do?

- Recap on **good programming practices**.
- Recap **on Python basics**.
- **Bootcamp on Git** with GitHub.

Good Programming Practices

The good programmer shalt

1. **INDENT** (consistently)
2. **USE MEANINGFUL NAME** (they make sense to anybody, you can read them)
3. CODE **SMALL FUNCTIONS**, DOING ONE THING (no unwanted side effects)
4. **COMMENT** SENSIBLY (if it is difficult to comment… rewrite code)
5. USE **EXCEPTIONS** RATHER THEN RETURN CODES
6. **TEST** EACH SINGLE LINE OF CODE
7. DRAW **SCHEMES** TO UNDERSTAND CODE
8. KNOW THE **LIBRARIES** OF CHOICE (you shall not invent from scratch)
9. BE **CONCURRENT**, WISELY
10. RE-READ, **REFINE**

limae labor et mora

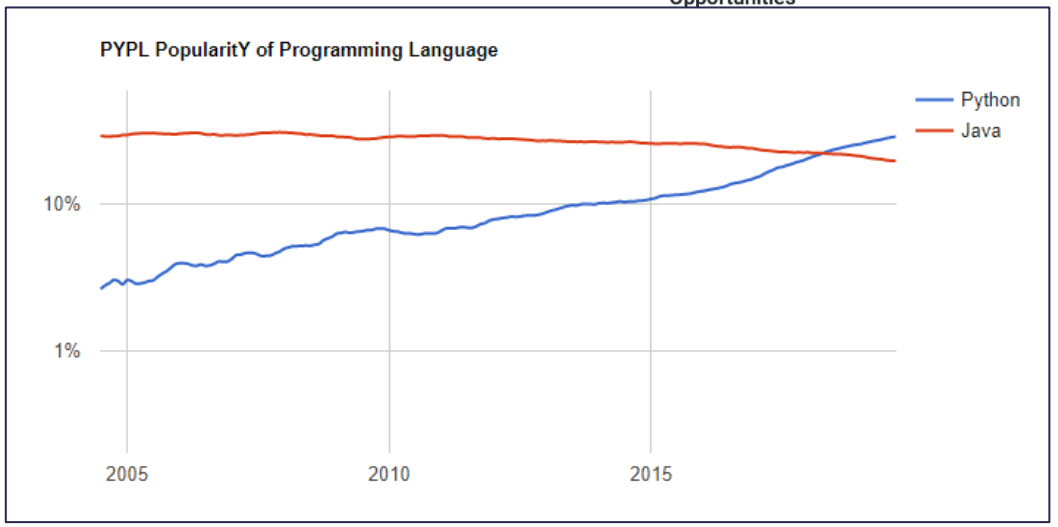
[Oratio, Ars Poetica, v. 291]

*filing work and thoughtful delay

Python

- Python is an amazingly versatile language.
- Most popular language in top universities.
- Language of choice for microservices, despite being slow.

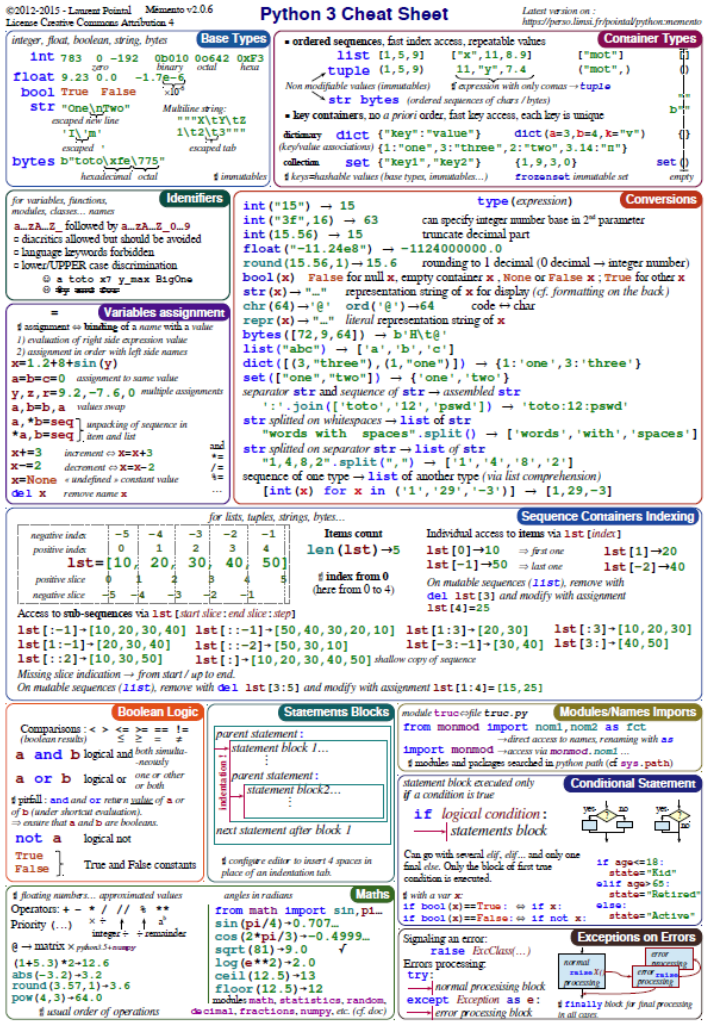
http://pypl.github.io/PYPL.html

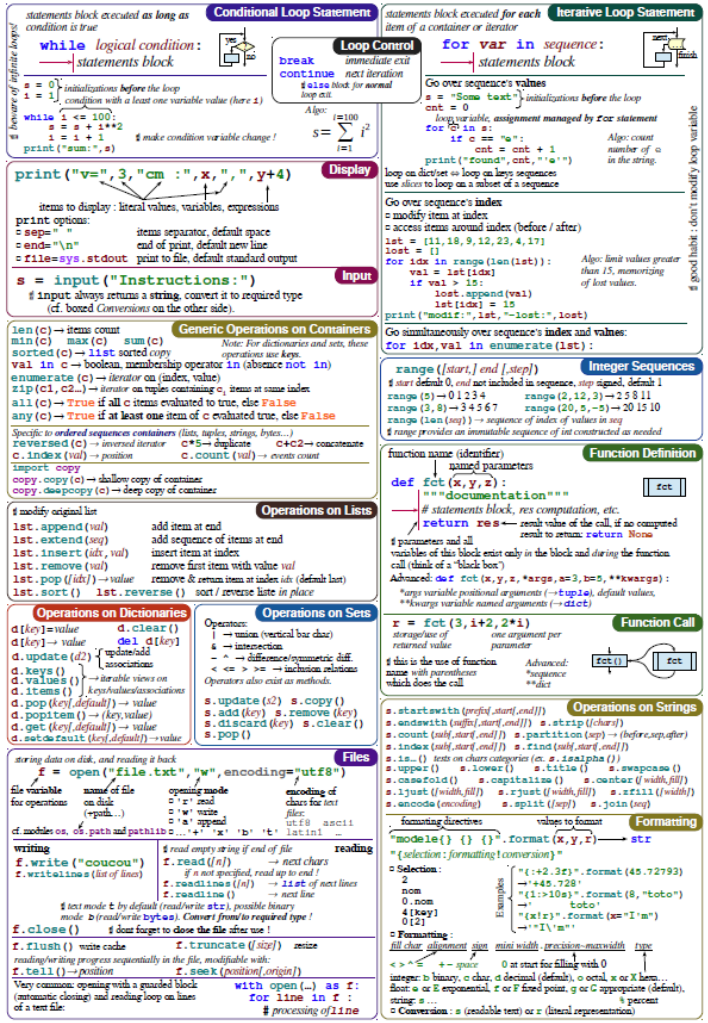


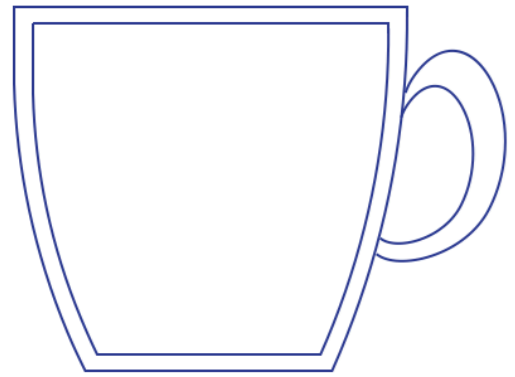
Worldwide, Sept 2021 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	29.48 %	-2.4 %
2		Java	17.18 %	+0.7 %
3		JavaScript	9.14 %	+0.8 %
4		C#	6.94 %	+0.6 %
5		PHP	6.49 %	+0.4 %
6		C/C++	6.49 %	+0.9 %

Python3 Recap







Hands-on Python Recap

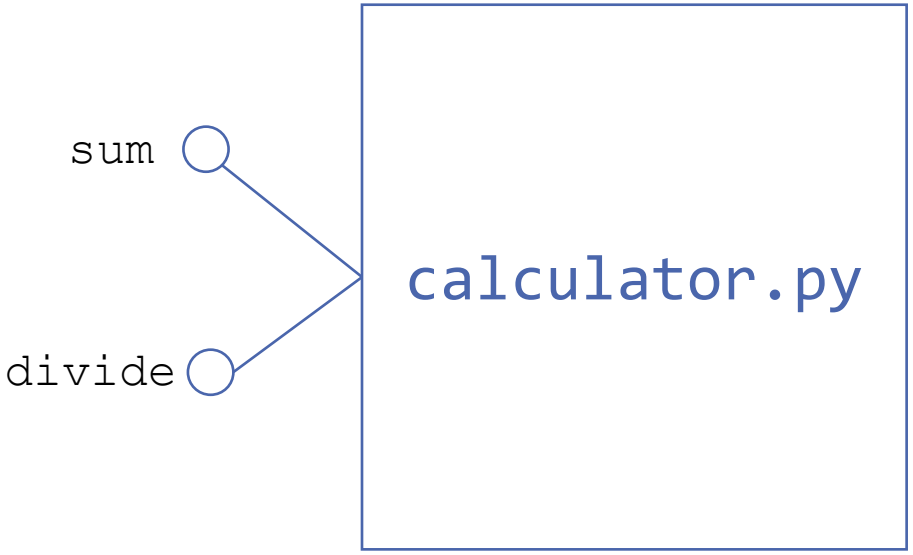
- Your customer wants a Python **calculator module**.
- You agreed on a **specification** and sketched some **interfaces**.
- You should submit a **prototype** by the end of next hour, capable of performing **all 4 operations**.

Interface

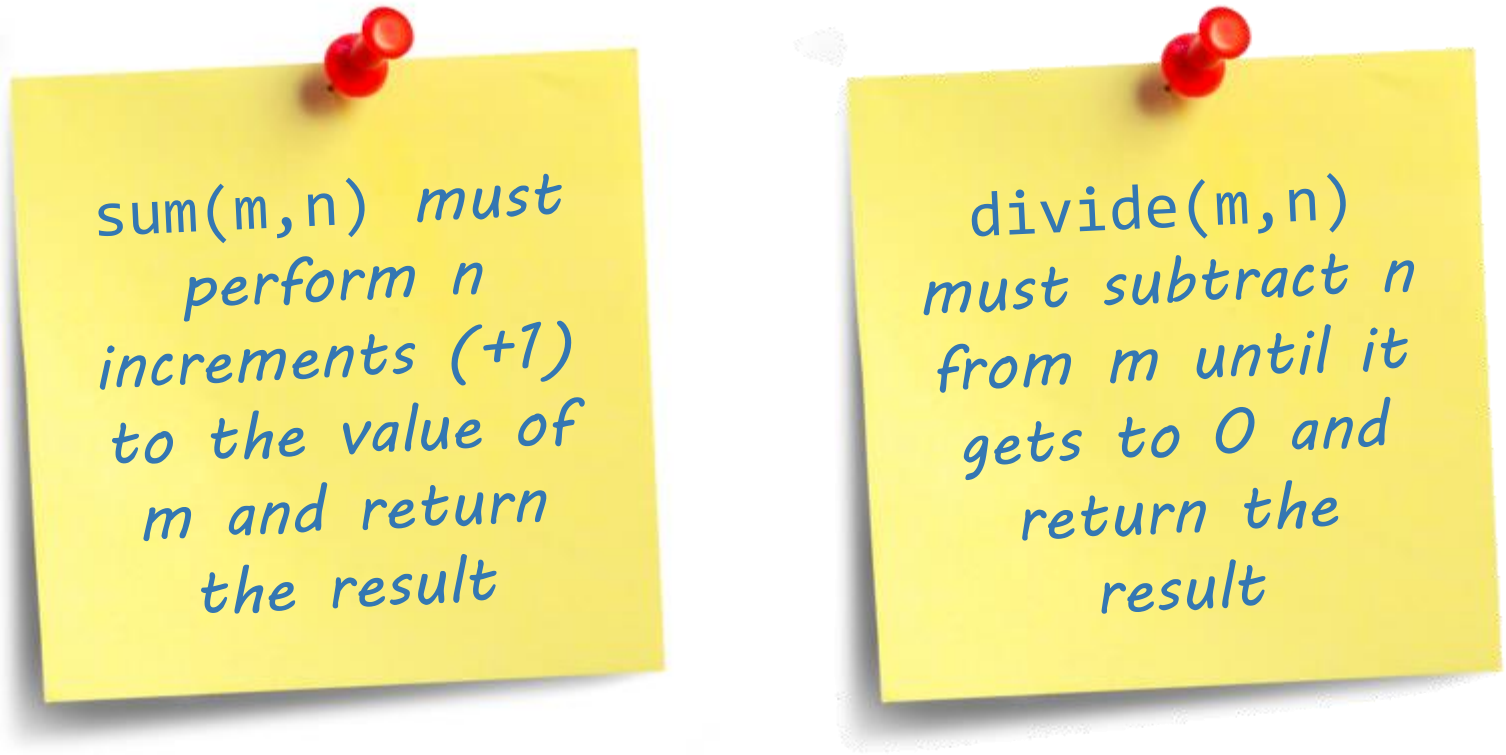
An **interface** is a *contract* that spells out how a certain software component interacts with the rest of the world. Python does not feature `interfaces` as Java and C#. However, programmers can always agree on an API, which does not bind to an implementation and stick to it.

```python
#calculator.py

def sum(m, n):
    #TO DO

def divide(m, n):
    #TO DO
```

https://en.wikipedia.org/wiki/Duck_typing



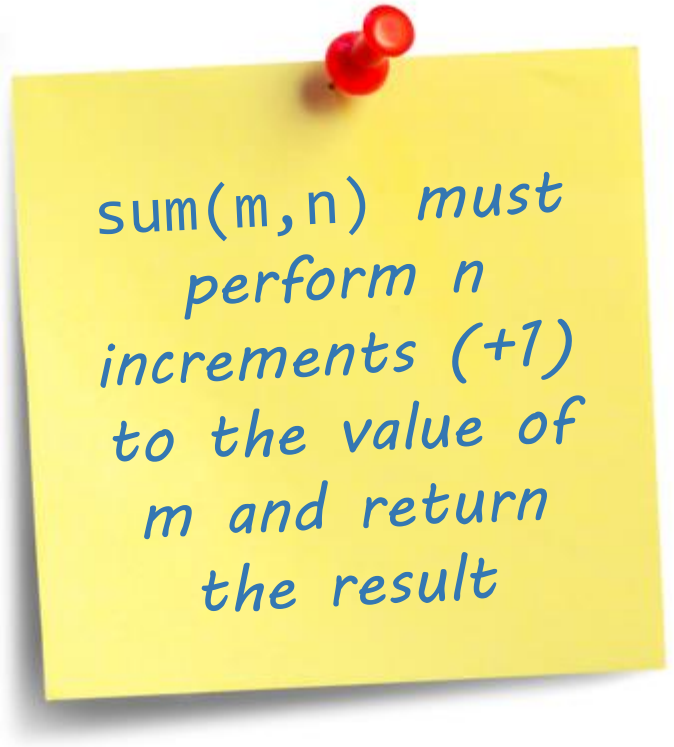
Requirements Specification



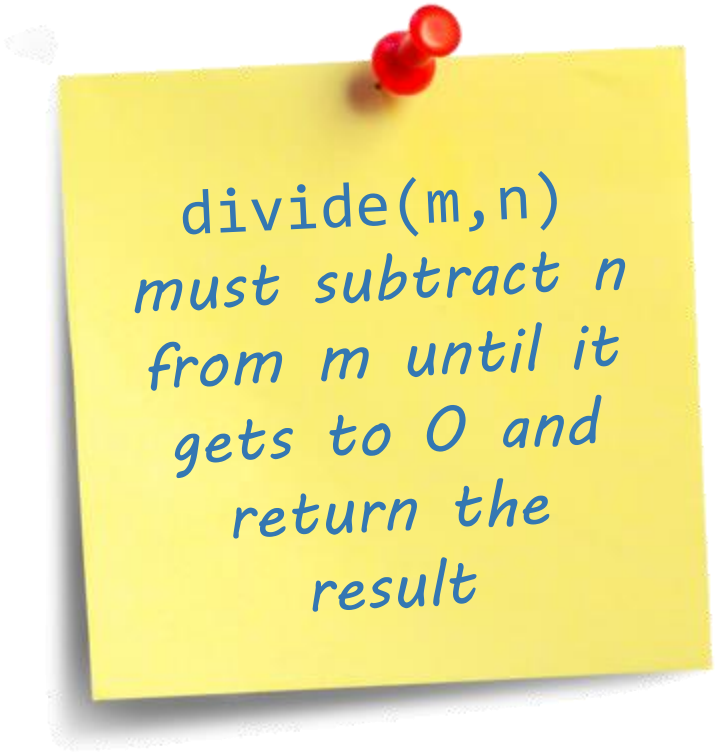
We start by writing a `calculator.py` module.
The module must work on integers in $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$
Then, we will built a Calculator class by exploiting it.

Sum

```
def sum(m, n):
    result = m
    if n < 0:
        for i in range(abs(n)):
            result -= 1
    else:
        for i in range(n):
            result += 1
    return result
```



Divide




```python
def divide(m, n):
    result = 0
    negativeResult = m > 0 and n < 0 or m < 0 and n > 0
    n = abs(n)
    m = abs(m)

    while (m - n >= 0):
        m -= n
        result+=1

    result = -result if negativeResult else result

    return result
```

Make a Class

```
import calculator as c

class FooCalculator:

    #empty constructor
    def __init__(self):
        pass

    def sum(self, m, n):
        return c.sum(m,n)

    def divide(self, m, n):
        return c.divide(m,n)
```

Set up Git

- Git is an open-source version control system (VCS).
- Open GitBash (or bash).
- Set a Git username:

```
Stefano@DESKTOP-BG5566C MINGW64 ~
$ git config --global user.name "Stefano"

Stefano@DESKTOP-BG5566C MINGW64 ~
$ git config --global user.name
Stefano
```

- Analogously, set up your `user.email`.

Create a Repo

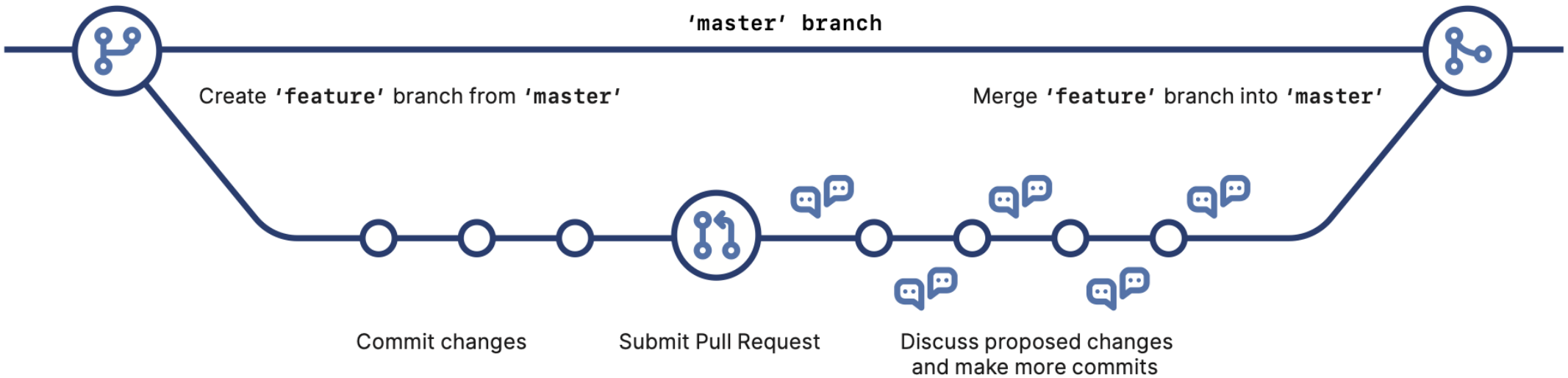
- Go to github.com and enter with your credentials.
- Repositories are the place where your projects live.
- In the upper-right corner of any page, click **+** and then **New Repository**.
- Type a short, memorable name, e.g. `ase-fall-20`.
- This repo will be **Public**.
- Initialise it with a **README**.
- Click **Create a repository**.

Clone your Repo

- Open GitBash (or bash).
- Move to the directory where you want to store your work (e.g., `ASE`).
- Use the command `git clone [your repo url]`.
- Create a new folder named `Lab_1` and move there the `Calculator` project folder. Then:

```
git add *
git commit -m "first commit"
git push
```

GitHub Flow



https://guides.github.com/introduction/flow/

Fork your mate’s repo

- Now, suppose you want to implement some peer review policy to your friend’s code.
- You can fork his/her repo.
- Work locally on his/her code to improve it or fix bugs.
- Submit a pull request to the project owner.

A **fork** is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.

How to? (1)

```
git remote -v origin
https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch) origin
https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
```

- Ask your neighbour for his/her GitHub nickname.
- Go to his/her repo and click on **Fork** (top right corner).
- A new repo has been added to YOUR GitHub account. Clone it in a folder **different** from the one of your project.
- When you fork a project in order to propose changes to the original repository, you can configure Git to pull changes from the original, or *upstream*, repository into the local clone of your fork.
- Type `git remote -v` and press Enter. You'll see the current configured remote repository for your fork.

How to? (2)

- Type `git remote add upstream`, and then paste the URL of your mate's repo and press Enter. It will look like this:

```
git remote add upstream <original repo url>
```

- Try again with `git remote -v`

```
$ git remote -v
origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
upstream  https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (fetch)
upstream  https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (push)
```

Sync a Repo (1)

- Sync a fork of a repository to keep it up-to-date with the upstream repository.
- Before you can sync your fork with an upstream repository, you must configure a remote that points to the upstream repository in Git.

```
$ git fetch upstream
remote: Counting objects: 75, done.
remote: Compressing objects: 100% (53/53), done.
remote: Total 62 (delta 27), reused 44 (delta 9)
Unpacking objects: 100% (62/62), done.
From https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY
 * [new branch]      master     -> upstream/master
```

Sync a Repo (2)

- Check out your fork's local master branch.

```
$ git checkout master
Switched to branch 'master'
```

- Merge the changes from upstream/master into your local master branch. This brings your fork's master branch into sync with the upstream repository, without losing your local changes.

```
$ git merge upstream/master
Updating a422352..5fdff0f
Fast-forward
 README                    |    9 -------
 README.md                 |    7 ++++++
 2 files changed, 7 insertions(+), 9 deletions(-)
 delete mode 100644 README
 create mode 100644 README.md
```

Unit Testing

- Unit testing involves verifying that each software unit meets its specification.
- Your customer's IT centre has written some unit tests for the implemented functionalities (functions) of the `FooCalculator`.

- Download the **tests** from the Moodle or from [http://pages.di.unipi.it/forti/ase/tests.zip]
- Test the repo you forked by running them.

A moment of wisdom

- Try to implement changes to the forked repo so to correct any possible bug.
- Enter your mate's repo on GitHub and send a **Pull Request**, once the code you have passes all unit tests.

Pull requests let you tell others about changes you've pushed to a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before the changes are merged into the repository.

Debugging!

```
def divide(m, n):
    result = 0
    negativeResult = m > 0 and n < 0 or m < 0 and n > 0
    n = abs(n)
    m = abs(m)

    if n == 0:
        raise ZeroDivisionError('You cannot divide by 0!')

    while (m - n >= 0):
        m -= n
        result+=1

    result = -result if negativeResult else result

    return result
```

In-class work

- Add other two functionalities to (your implementation of) the `Calculator`.
- Extends unit tests accordingly.
- Commit to GitHub when tests are passed.

`subtract(m,n)` must perform n decrements (-1) to the value of m and return the result

`multiply(m,n)` must sum the value of m for n times and return the result

`gcd(m,n)` must implement Euclid's Algorithm