



C O M P A S

```
def smooth_mesh_length(mesh, lmin, lmax, fixed=None, kmax=100):
    """
    Smooth the mesh by iteratively adjusting the lengths of the edges.
    The function iterates over the edges of the mesh, adjusting their
    lengths until they are within the specified range [lmin, lmax].
    The iteration stops when the number of edges that need to be
    adjusted is less than kmax.

    Parameters
    ----------
    mesh : Mesh
        The mesh to be smoothed.
    lmin : float
        The minimum allowed edge length.
    lmax : float
        The maximum allowed edge length.
    fixed : set
        A set of edges that are fixed and should not be adjusted.
    kmax : int
        The maximum number of edges that need to be adjusted.

    Returns
    -------
    mesh : Mesh
        The smoothed mesh.
    """
    if not callable(callback):
        raise Exception('Callback is not callable.')

    fixed = fixed or []
    fixed = set(fixed)

    for k in range(kmax):
        # Update the mesh
        attr = mesh.vertex[key]
        attr['x'] += d * (c[0] - p[0])
        attr['y'] += d * (c[1] - p[1])
        attr['z'] += d * (c[2] - p[2])

        # If callback:
        callback(mesh, k, callback_args)

    return mesh
```

Building blocks

Code management

Python programming

COMPAS programming

Building **blocks**







Complexity  
Isolation

Economies  
of Scale

# Building blocks in Python

```
def awesome_convex_hull(points):  
    # Insert here magical  
    # convex hull algorithm  
  
    return faces
```

Stand-alone script

hull.py

File

Module

hull.py

File



Package

`compas.geometry`

Folder

Module

`hull.py`

File

Library

Package

Module

compas

compas.geometry

hull.py

Installable folders

Folders

File

Developer's laptop

User's laptop

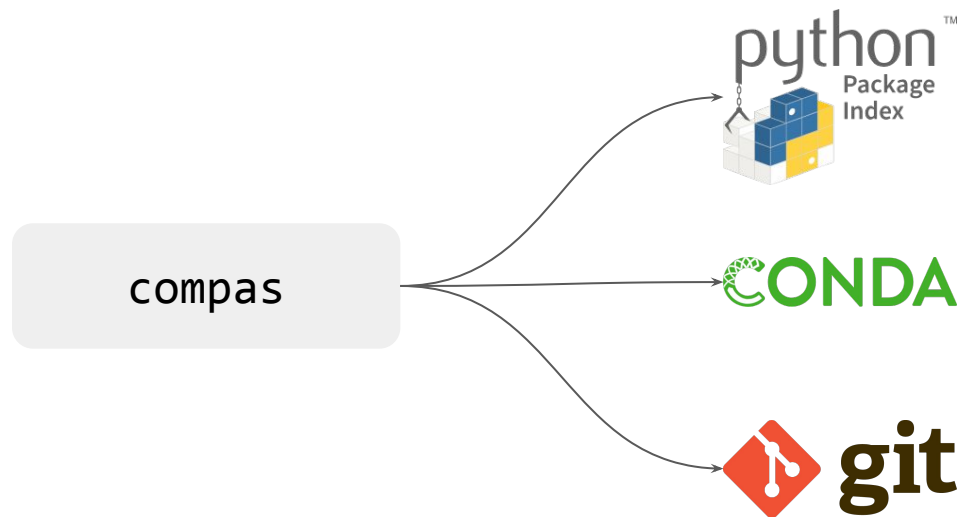
compas

?



The diagram illustrates a data transfer process. On the left, under the label 'Developer's laptop', there is a light gray rounded rectangle containing the text 'compas'. A horizontal dashed blue line with a solid blue arrowhead at its right end extends from the right side of this rectangle towards the right. On the right, under the label 'User's laptop', there is a light gray circle containing a large black question mark. The arrow points from the 'compas' box to the question mark circle, indicating the direction of data flow.

Developer's laptop

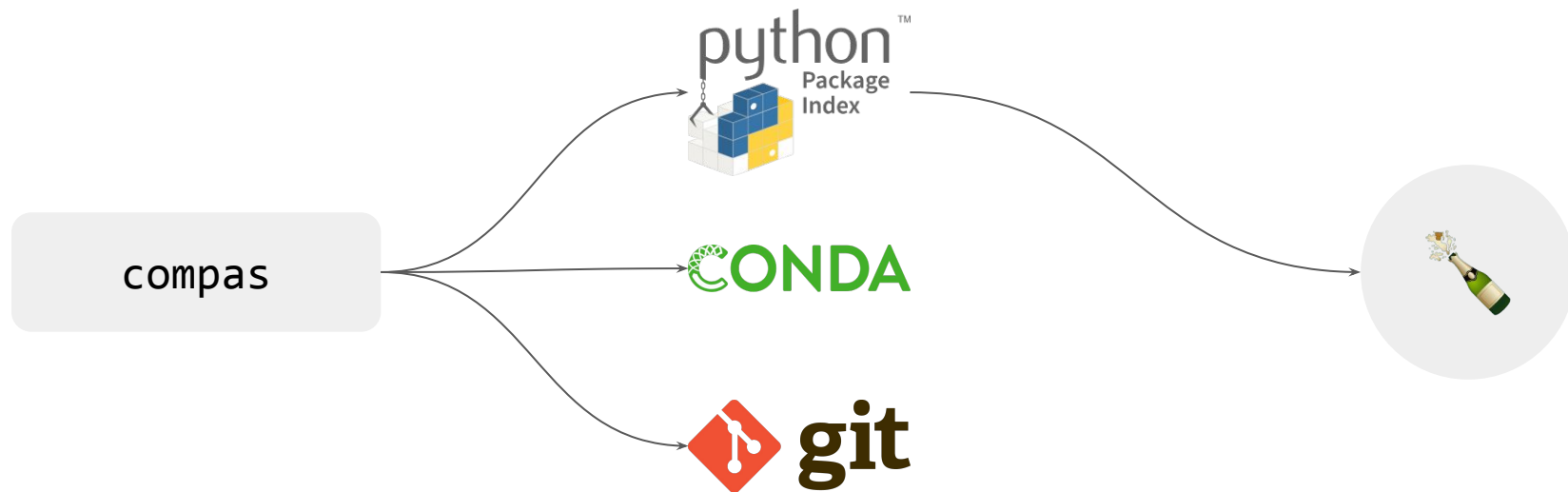


User's laptop



Developer's laptop

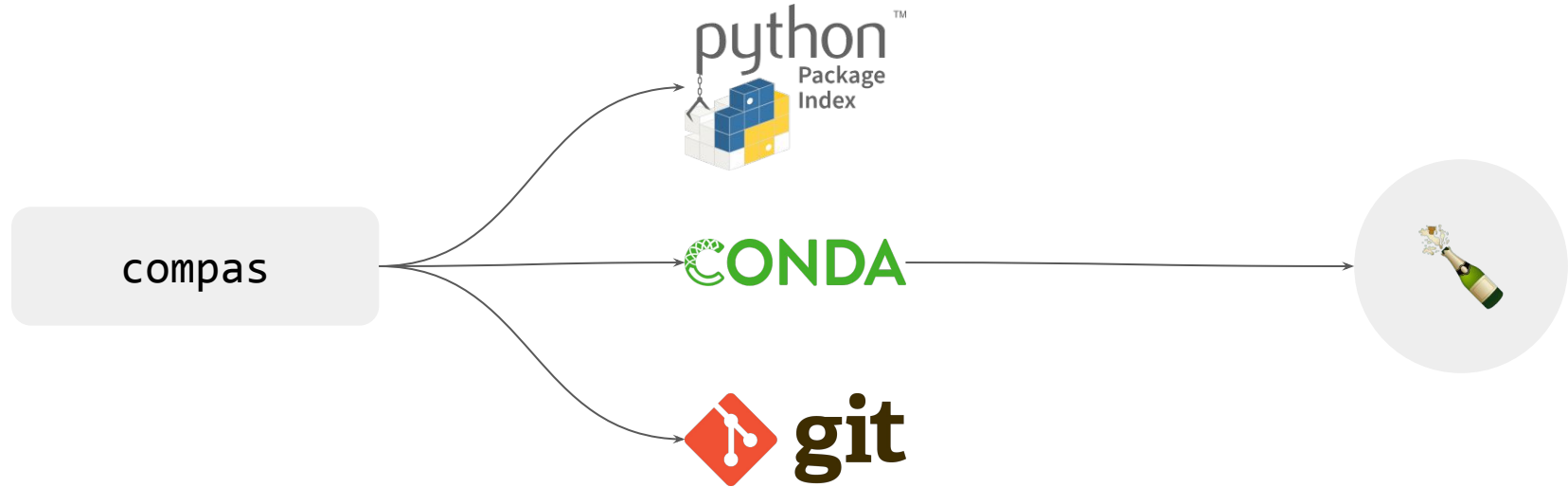
User's laptop



```
pip install compas
```

Developer's laptop

User's laptop

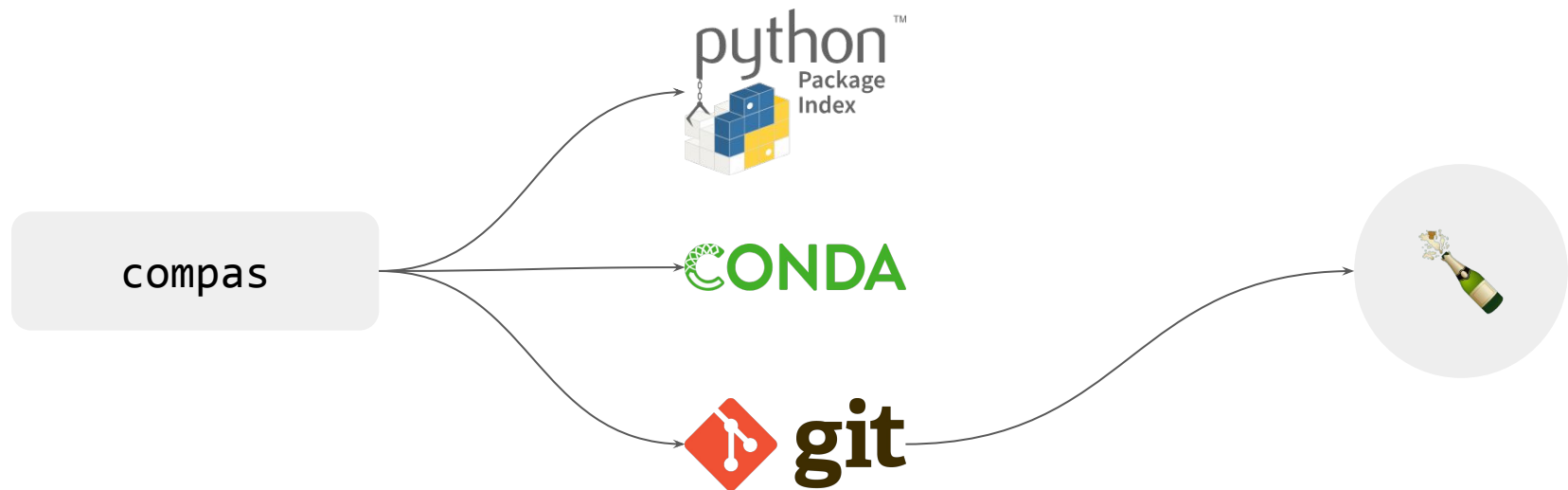


```
conda install compas
```



Developer's laptop

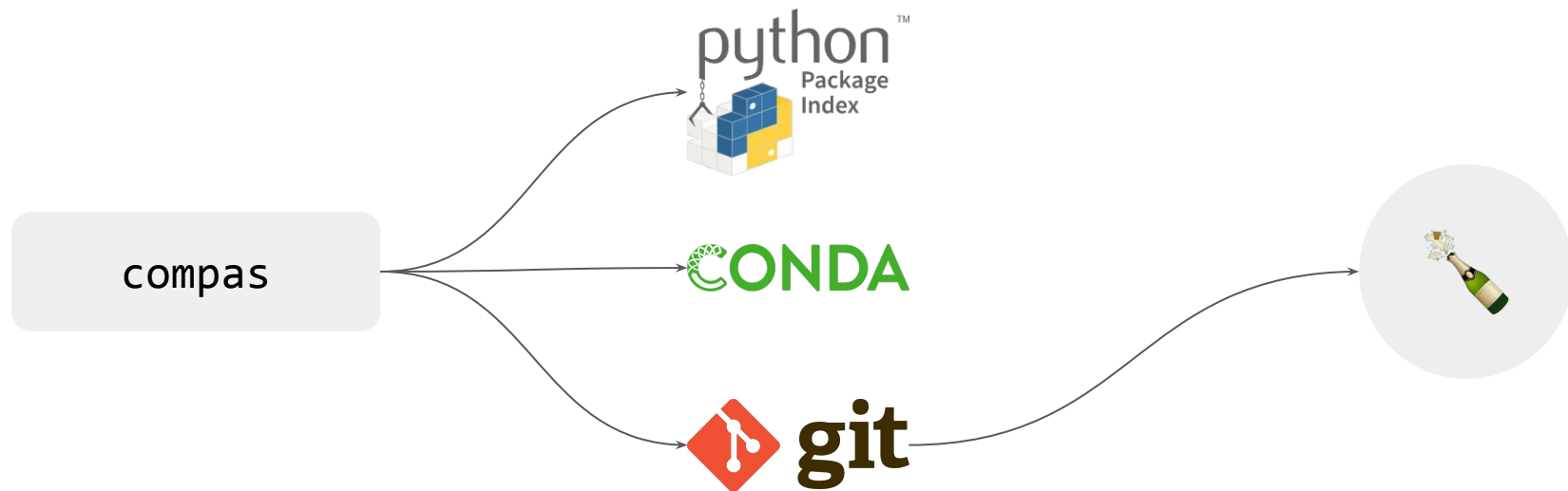
User's laptop



```
pip install git+https://github.com/compas-dev/compas.git
```

Developer's laptop

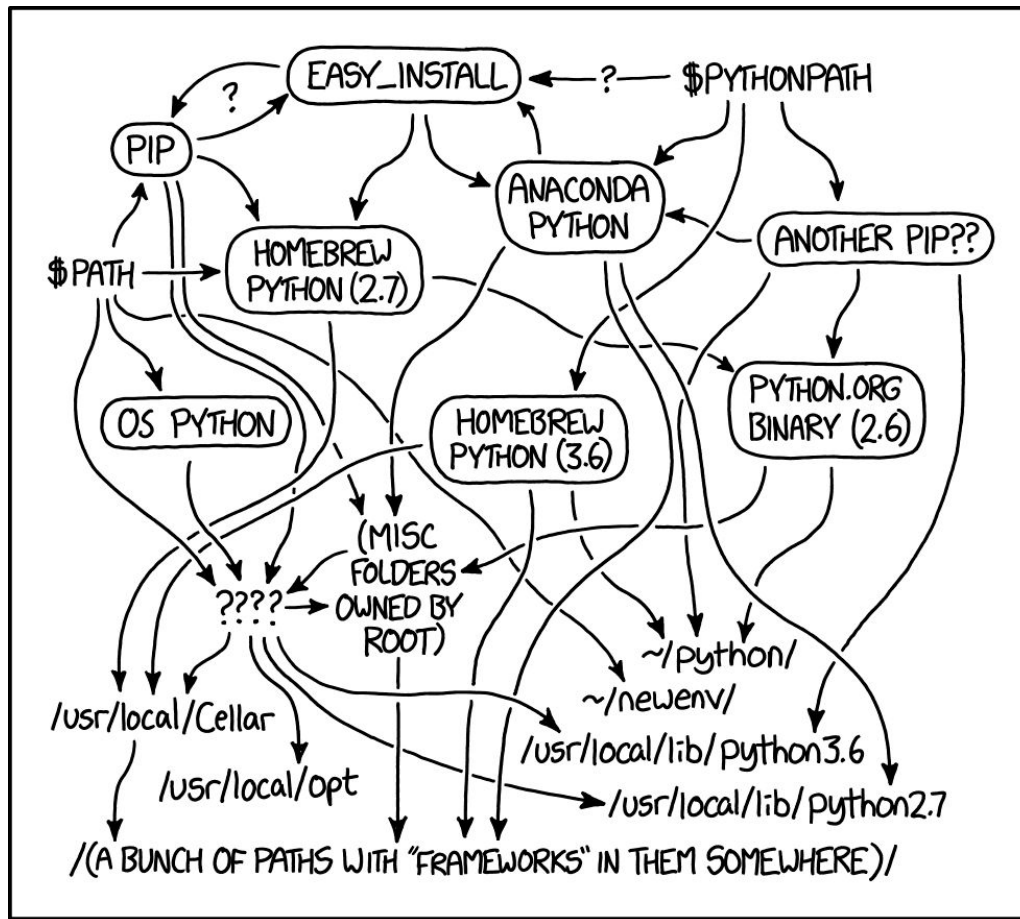
User's laptop



```
pip install -e .
```

“I’ll install into **base env**, *it’ll be fine*”

-A formerly happy user



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED  
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.



User's laptop with virtual environments

ita19

python=3.6

compas=0.8



User's laptop with virtual environments

ita19

python=3.6

compas=0.8

ita19-dev

python=3.6

compas@master

my-slack-bot

python=3.8

slackclient=..





User's laptop with virtual environments

ita19

python=3.6

compas=0.8

ita19-dev

python=3.6

compas@master

my-slack-bot

python=3.8

slackclient=..

CPython



IronPython

CPython  $\neq$  IronPython



User's laptop with virtual environments

ita19

python=3.6

compas=0.8

ita19-dev

python=3.6

compas@master

my-slack-bot

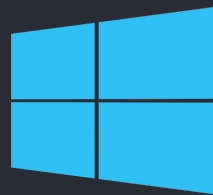
python=3.8

slackclient=..



```
python -m compas_rhino.install -v 6.0
```

⌘ + SPACE



Terminal

Anaconda Prompt



```
(base) conda activate ita19  
(ita19) pip -m compas_rhino.install -v 6.0
```

Don't type this part!





Code **management**



# Git



Working directory

# Git

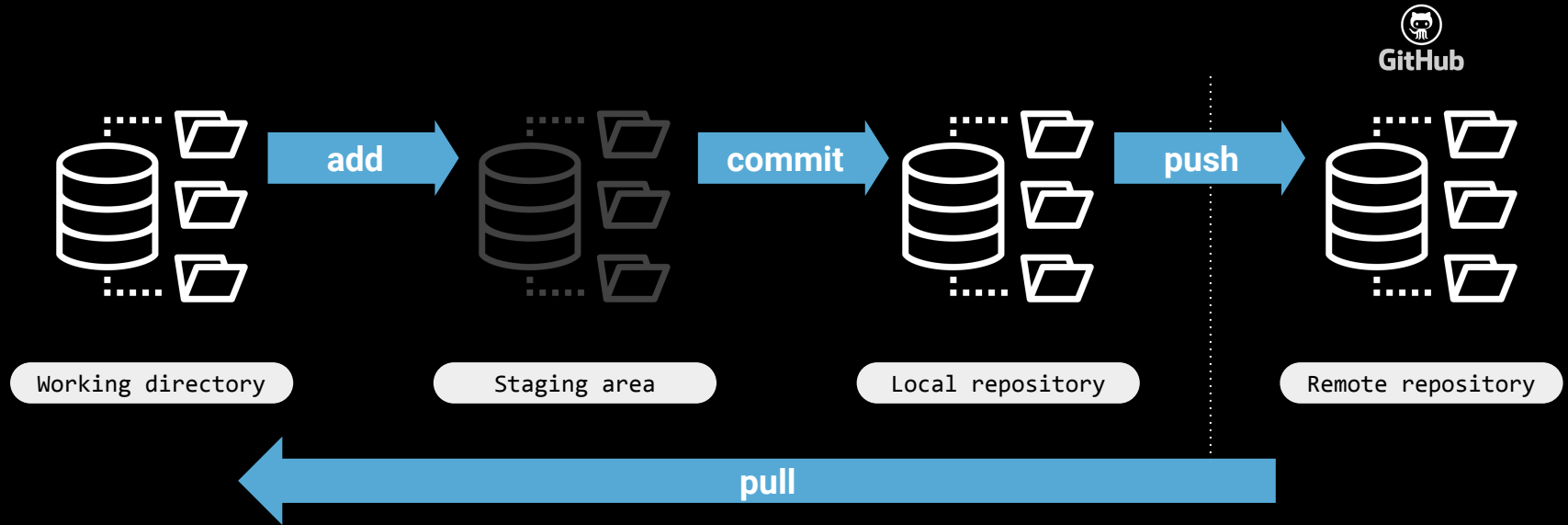


Working directory



Remote repository

# Git



Better **personal workflow**  
Friction-less **collaboration**

# Better personal workflow

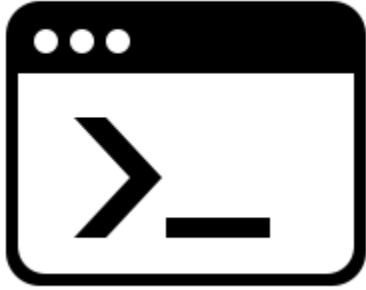
- Entire local history of changes
- Ability to jump back and forth in time
- Removes file clutter
- Add extra safety and traceability

Code management

# Friction-less collaboration

- Branching and merging is simple
- Allows async collaboration
- Improves project visibility

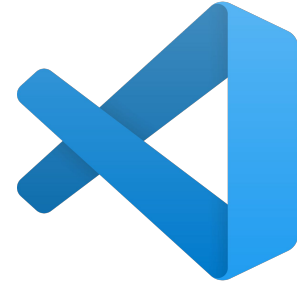




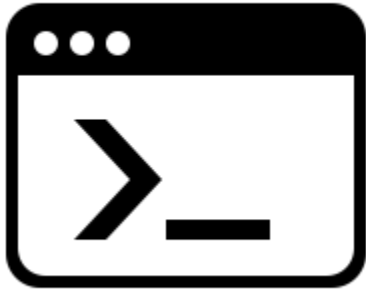
Command line



Visual Client  
*e.g. SourceTree,  
GitHub Desktop*



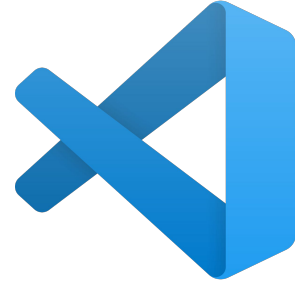
IDE Integration  
*e.g. VS Code*



Command line



Visual Client  
*e.g. SourceTree,  
GitHub Desktop*



IDE Integration  
*e.g. VS Code*

Download **at least one** of them

# Let's do a simple exercise

1. Clone **compas** repository
2. Create a new **conda environment**
3. Install **compas** from source code

compas-dev / compas

Unwatch 14 Star 73 Fork 33

Code Issues 22 Pull requests 3 Projects 1 Wiki Security Insights Settings

compas main library <https://compas-dev.github.io/main/>

Edit

Manage topics

2,340 commits 6 branches 78 releases 1 environment 18 contributors MIT

Branch: master New pull request

Create new file Upload files Find file Clone or download

gonzalocasas	Merge pull request #351 from tetov/patch-1
.github	Create FUNDING.yml
bin	Rework the symlink polyfill so that it only asks escalation once
data	expand test_mesh.py
docs	Merge pull request #346 from tetov/fix-callbacks-tut
src	Small spelling mistake in docstring
tests	tests fixed

Clone with HTTPS Use SSH

Use Git or checkout with SVN using the web URL.

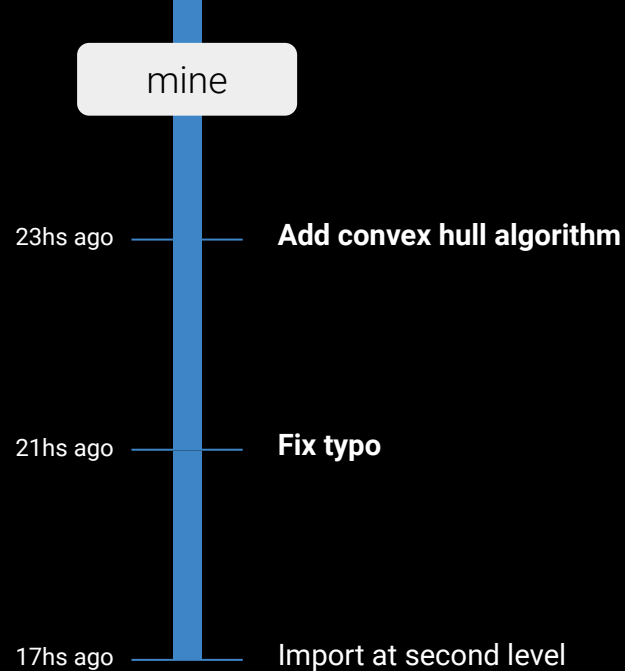
<https://github.com/compas-dev/compas.git>

Open in Desktop Download ZIP

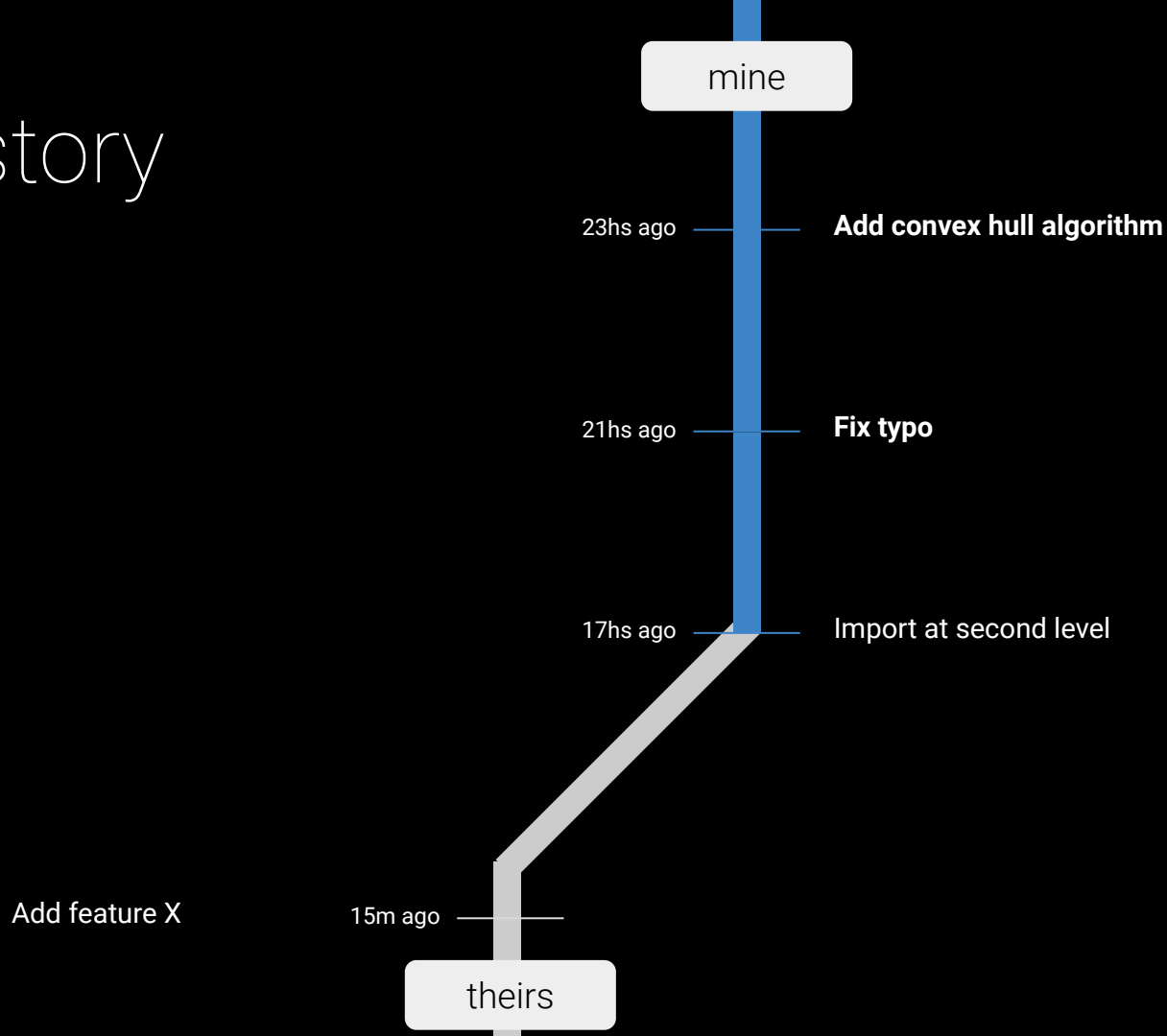


# Collaborative workflows

# Code history



# Code history





# Code history

## Branches

Add feature X

15m ago

theirs

30m ago

**Add unit tests**

17hs ago

**Import at second level**

21hs ago

**Fix typo**

23hs ago

**Add convex hull algorithm**

mine

# Code history

## Merge

Add feature X

15m ago

theirs

17hs ago

Import at second level

30m ago

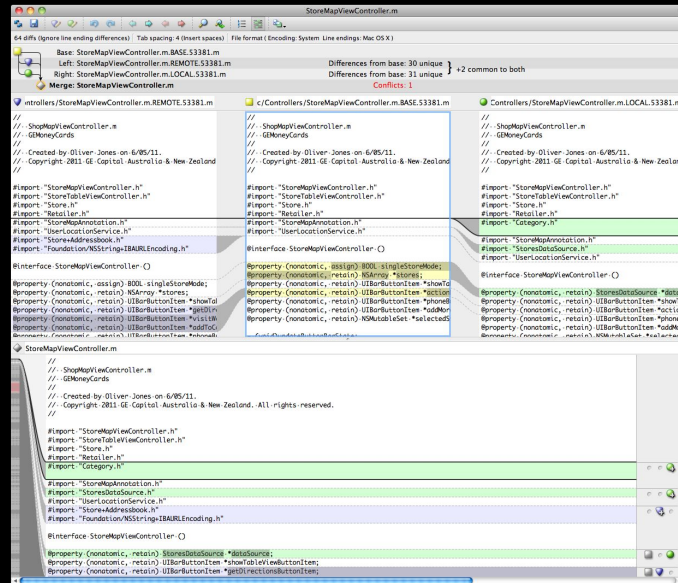
**Add unit tests**

5m ago

**Merge**

mine

# Use a visual merge tool



P4Merge

# Issue management

## Reporting issues

- File bugs, feature requests, enhancements as **Github Issues**
- Describe them properly:  
*Single-sentence reports are worthless*
- Add issue templates (for your repos)
- Ideally, add test demoing the error

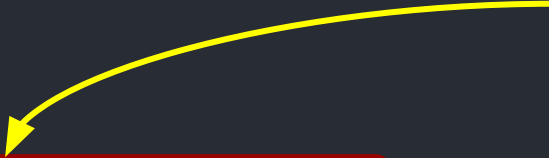
## Fixing issues

- Refer to it in the commit message  
eg. `fix #42: resolving age-old question`
- Add tests to avoid regressions

**Question** instead of issue?  
Maybe a **Forum** is best



Change to match  
your selected folder!



```
cd path/to/your/clone/of/compas/compas
conda activate ita19-dev
pip install -e .
python
```

# **Python** programming

Data containers

Control flow

Best practices



# Data containers

- Basic data types
- Lists vs dictionaries vs sets vs tuples
- OOP: Classes & objects

examples

```
value = 42
```

```
name = 'Adam'
```

```
years = [79, 80, 82, 84, 92]
```

```
point = (10, 20, 15)
```

```
{'x': 10, 'y': 20, 'z': 15}
```

```
bunny = Mesh()
```

# Data containers

- Mutability: by ref vs by value
- Context managers
- File operations
- Decorators

examples

```
with Serial() as serial  
    # do something:
```

```
with open('filepath', 'r') as f:  
    f.readlines()
```

```
@property  
def name(self):  
    return self._name
```

# Control flow

- Conditionals & loops
- Function calls & recursion
- Unhappy code paths
- Generators
- Variable scope & shadowing
- Callbacks

examples

```
if value >= 42: print('ok')
```

```
for i in 'hi ITA': print(i)
```

```
def calculate_answer(q):
```

```
def factorial(n):
```

```
try..except..finally
```

Always **reduce** complexity

# Which complexity?

**ALGORITHMS**

+

**DATA STRUCTURES**

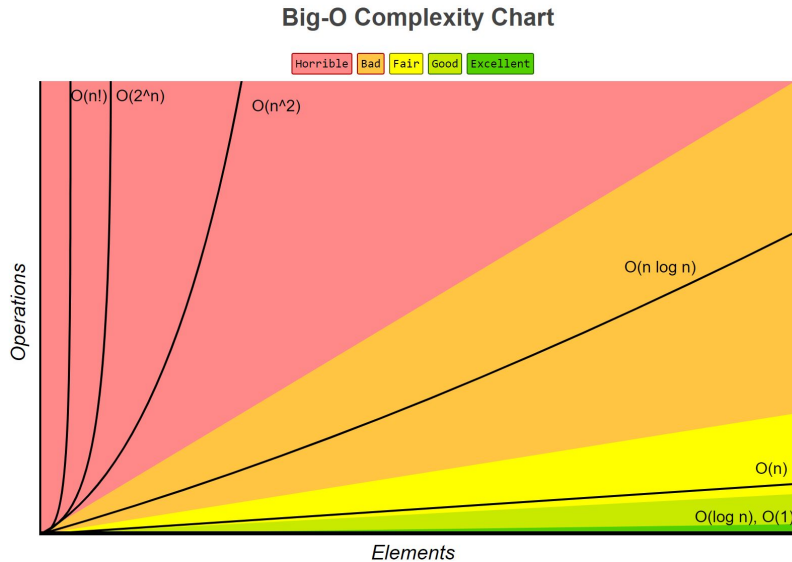
=

**PROGRAMS**

time & space complexity

code complexity

# Time & space complexity



Source: <http://www.bigocheatsheet.com>

- Big-O notation
- Quantifies amount of time and space required in relation to the input

Time complexity:  $O(n)$

```
def search(items, target):  
    for i in range(len(items)):  
        if items[i] == target:  
            return i  
    return -1
```

Time complexity:  $O(n^2)$

```
def has_duplicates(items):  
    for outer in range(len(items)):  
        for inner in range(len(items)):  
            if inner == outer: continue  
  
            if items[inner] == items[outer]:  
                return True  
  
    return False
```



# Code complexity

- **Lots of metrics:**
  - Cyclomatic complexity
  - Line count 🧑
  - Object coupling
  - Class hierarchy level
  - Method cohesion
  - etc.

But no definitive model to quantify it.

# Code complexity: strategies

- **Always optimize for readability**
- Follow naming & style conventions (PEP8)
- Naming is hard, take your time
- Encapsulate complexity: solve it just once
- Don't reinvent the wheel, reuse encapsulated code
- Less code is always more
- Leverage tools to assist your coding (linters, formatters, static analyzers, etc)
- Refactor continuously

# Coding and style conventions

- **PEP8**: Official style guide for Python
- Conventions are not spaces vs tabs, are about **signal versus noise**
- Use linters, formatters, config files to auto-enforce conventions
- Use project templates using cookiecutter

# Zen of Python

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.
8. Special cases aren't special enough to break the rules.
9. Although practicality beats purity.
10. Errors should never pass silently.
11. Unless explicitly silenced.
12. In the face of ambiguity, refuse the temptation to guess.
13. There should be one-- and preferably only one --obvious way to do it.
14. Although that way may not be obvious at first unless you're Dutch.
15. Now is better than never.
16. Although never is often better than \*right\* now.
17. If the implementation is hard to explain, it's a bad idea.
18. If the implementation is easy to explain, it may be a good idea.
19. Namespaces are one honking great idea -- let's do more of those!

- PEP 20: The Zen of Python
- 20 aphorisms / guidelines

```
import this
```

# Principles

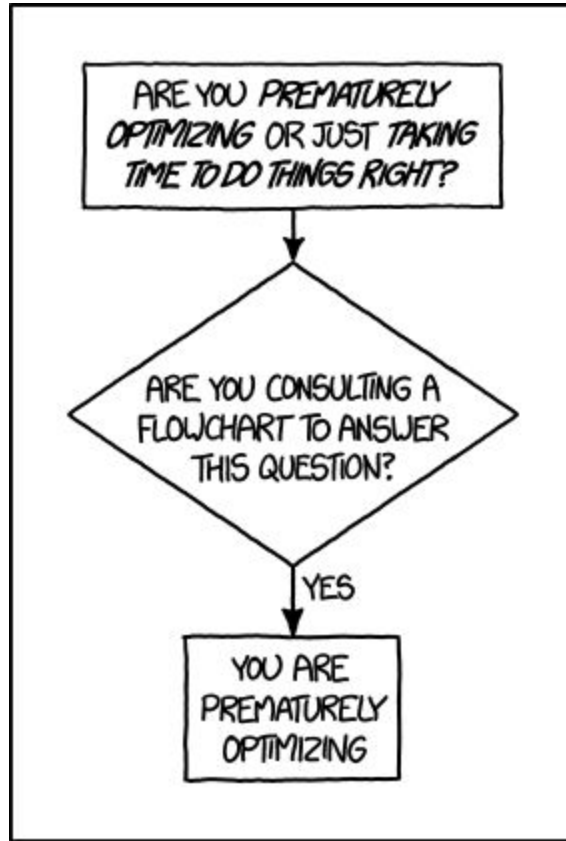
- **DRY:** Don't repeat yourself
- **YAGNI:** You ain't gonna need it
- **KISS:** Keep it simple stupid
- **SOLID:**
  - **S:** Single Responsibility Principle
  - **O:** Open-Closed Principle
  - **L:** Liskov Substitution Principle
  - **I:** Interface Segregation Principle
  - **D:** Dependency Inversion Principle

# Refactoring

- Refactoring is a state of mind
- Refactoring is a constant process
- Detecting “code smells”:
  - Duplicate Code
  - Long Method
  - Primitive Obsession
  - Speculative Generality
  - Premature Optimization
  - etc.

*“Changing software without altering external behavior but improving internal structure”*

-- Martin Fowler



# Design patterns

- **Blueprints**
- General, reusable solutions to a commonly occurring problem.

[github.com/faif/python-patterns](https://github.com/faif/python-patterns)

examples

Decorator

Singleton

Builder

Factory Method

Lazy evaluation

Circuit breaker

etc



Best practices

# Open source source

Use

Learn

Contribute

Publish



# Agile Manifesto

**Individuals and interactions** over processes and tools

**Working software** over comprehensive documentation

**Customer collaboration** over contract negotiation

**Responding to change** over following a plan

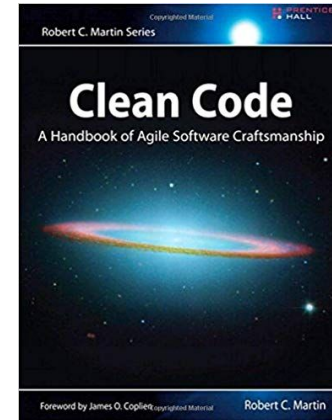
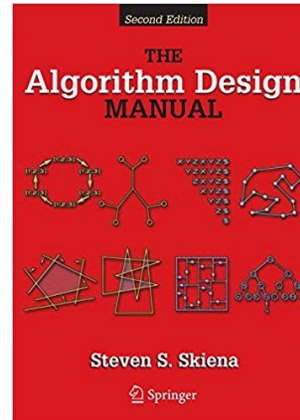
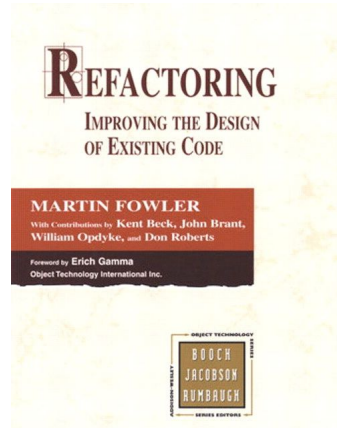
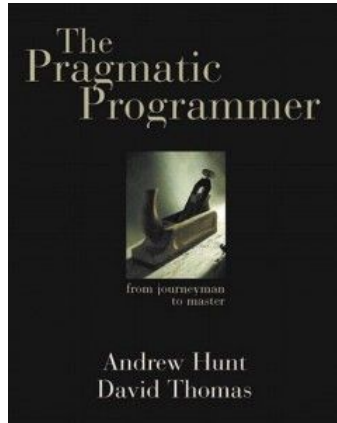
# Read other people's code

Pull Requests

Open Source

Pair Programming

# Reference material



**COMPAS** programming

# COMPAS programming

- Always import (max) second level
- Geometric keys
- Implementation-specific suffixes
- Using COMPAS inside Rhino
  - RPC
  - XFunc
  - Module unload
  - Script engine reset

# Slides & Assignment

[u.nu/ita19](https://u.nu/ita19)



# Join Slack



**<https://tinyurl.com/yxse82a7>**



