



# Python tutorial

Tips and tricks you may already know...

Prof. Maurício Girardi-Schappo (UFSC, Brazil)



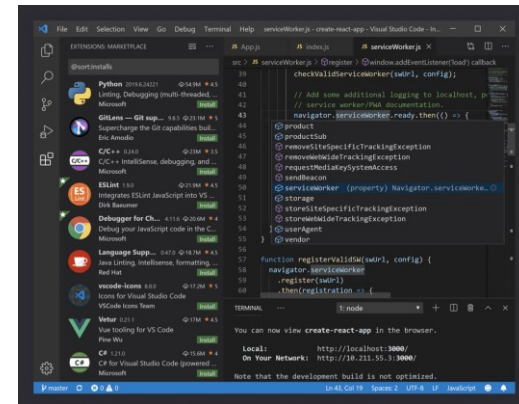
# Summary

- Coding:  
What is it really about?



- In the beginning...  
There was an IDE

- Python:  
must know



$$\begin{array}{l} A \rightarrow B \\ C \rightarrow D \\ E \rightarrow F \end{array}$$

# Disclaimer

- This lecture **is not about:**
  - Coding from the scratch
  - Advanced coding/python
- This lecture **is about:**
  - Giving a hands-on intro...



# Coding: what is it about?

- **Solving problems**  
you didn't know  
you had!

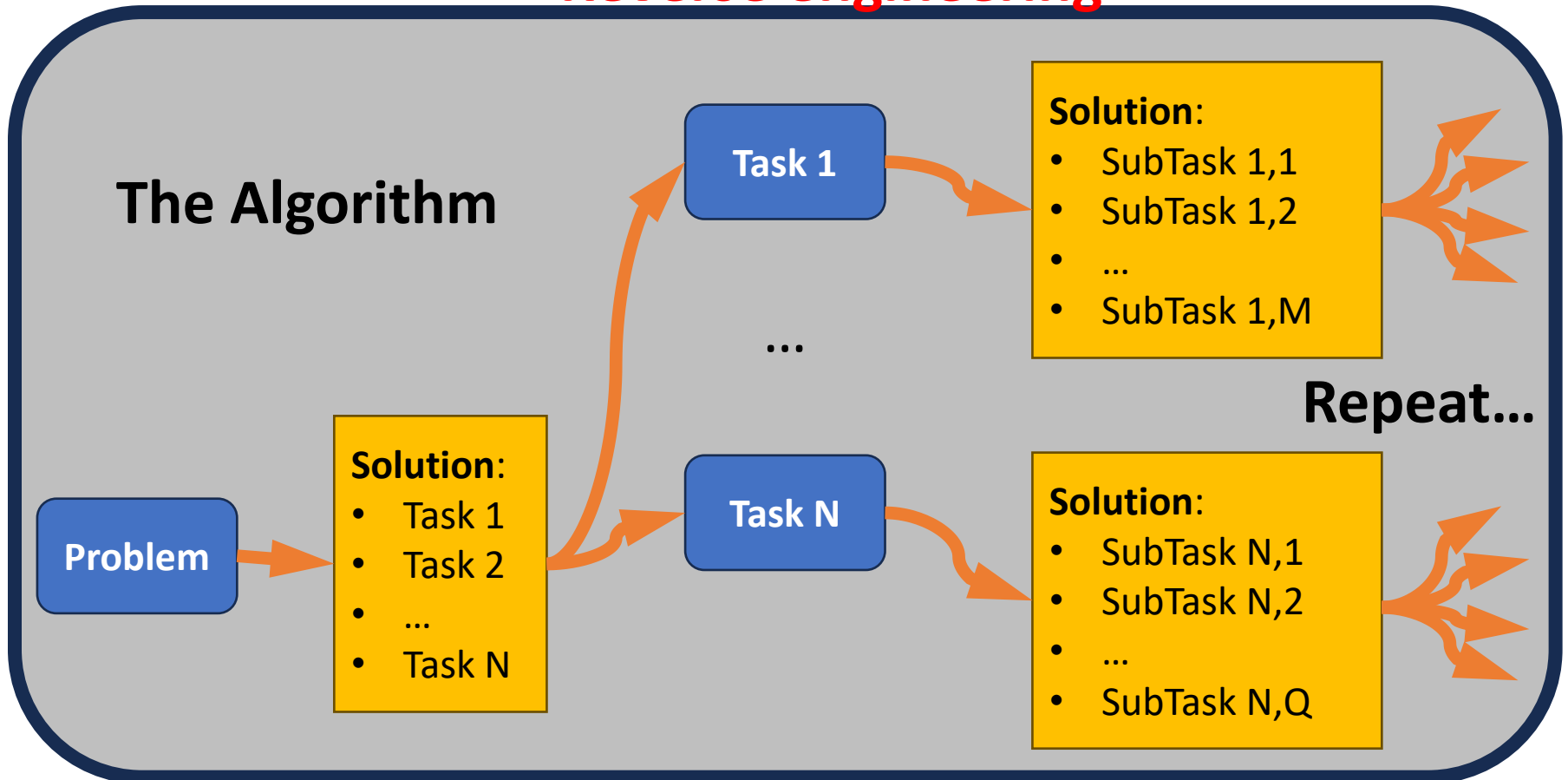


# Coding: what is it about?

- If you can make a sandwich,  
you **can code!** a.k.a.



**“Reverse engineering”**



# Coding: what is it about?

- If you can make a sandwich,  
you **can code**!



```
from kitchen import SandwichMaker, fridge

sandwich = SandwichMaker()

if sandwich.is_vegan():
    stuffing = fridge.get_available_vegs()
else:
    stuffing = fridge.get_everything()

sandwich.open() # we need to open the bread!
for item in stuffing:
    sandwich.add(item) # add stuff
sandwich.close() # and then close it!

grill(sandwich, temperature=150*Celsius)
eat(sandwich)
```

# Coding: elements...

- **Statements:**

Tells the “interpreter”  
what to do



**Each line  
here!**

E.g.:

- Create variables
- Check conditions
- Iterate over array
- Import stuff
- ...

```
from kitchen import SandwichMaker, fridge
```

```
sandwich = SandwichMaker()
```

```
if sandwich.is_vegan():  
    stuffing = fridge.get_available_vegs()  
else:  
    stuffing = fridge.get_everything()
```

```
sandwich.open() # we need to open the bread!  
for item in stuffing:  
    sandwich.add(item) # add stuff  
sandwich.close() # and then close it!
```

```
grill(sandwich, temperature=150*Celsius)  
eat(sandwich)
```



# Coding: elements...

- **Functions:**

Transform some values  
into other values



```
from kitchen import SandwichMaker, fridge

sandwich = SandwichMaker()

if sandwich.is_vegan():
    stuffing = fridge.get_available_vegs()
else:
    stuffing = fridge.get_everything()

sandwich.open() # we need to open the bread!
for item in stuffing:
    sandwich.add(item) # add stuff
sandwich.close() # and then close it!

grill(sandwich, temperature=150*Celsius)
eat(sandwich)
```



# Coding: elements...

- **Statements:**

Tells the “interpreter”  
what to do



**Each line  
here!**

E.g.:

- Create variables
- Check conditions
- Iterate over array
- Import stuff
- ...

```
from kitchen import SandwichMaker, fridge
```

```
sandwich = SandwichMaker()
```

```
if sandwich.is_vegan():  
    stuffing = fridge.get_available_vegs()  
else:  
    stuffing = fridge.get_everything()
```

```
sandwich.open() # we need to open the bread!  
for item in stuffing:  
    sandwich.add(item) # add stuff  
sandwich.close() # and then close it!
```

```
grill(sandwich, temperature=150*Celsius)  
eat(sandwich)
```

# Coding: elements...

- **Variables:**

names that hold values... (i.e., “boxes”)

They are  
assigned with  
the “=” operator

- int
- float
- str
- numpy.ndarray
- SandwichMaker



- **Types:**

Define what a variable  
can hold

- **Objects:**

Special variables  
that hold *abstract*  
**Types**

- **Classes:**

Statements that define  
*abstract* **Types**

```
from kitchen import SandwichMaker, fridge

sandwich = SandwichMaker()

if sandwich.is_vegan():
    stuffing = fridge.get_available_vegs()
else:
    stuffing = fridge.get_everything()

sandwich.open() # we need to open the bread!
for item in stuffing:
    sandwich.add(item) # add stuff
sandwich.close() # and then close it!

grill(sandwich, temperature=150*Celsius)
eat(sandwich)
```

# Coding: elements...

- Remember:

A **programming language**  
is a

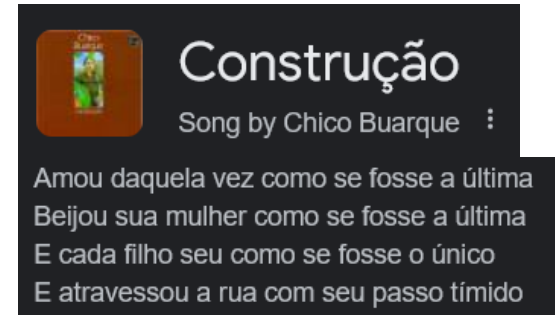
**Formal Language** not Poetry

i.e. no ambiguity; only “one way”  
of writing it; writing rules are  
strict (syntax is “absolute”)

- Errors will happen**

**Don't avoid** them, **read** and **understand**!

- ChatGPT/Google/Stackoverflow**  
can help. **Use with caution.**  
Use clever queries with proper  
“keywords”



```
1 "111" + 10

-----
TypeError                                Traceback (most recent call last)
Cell In[14], line 1
----> 1 "111" + 10

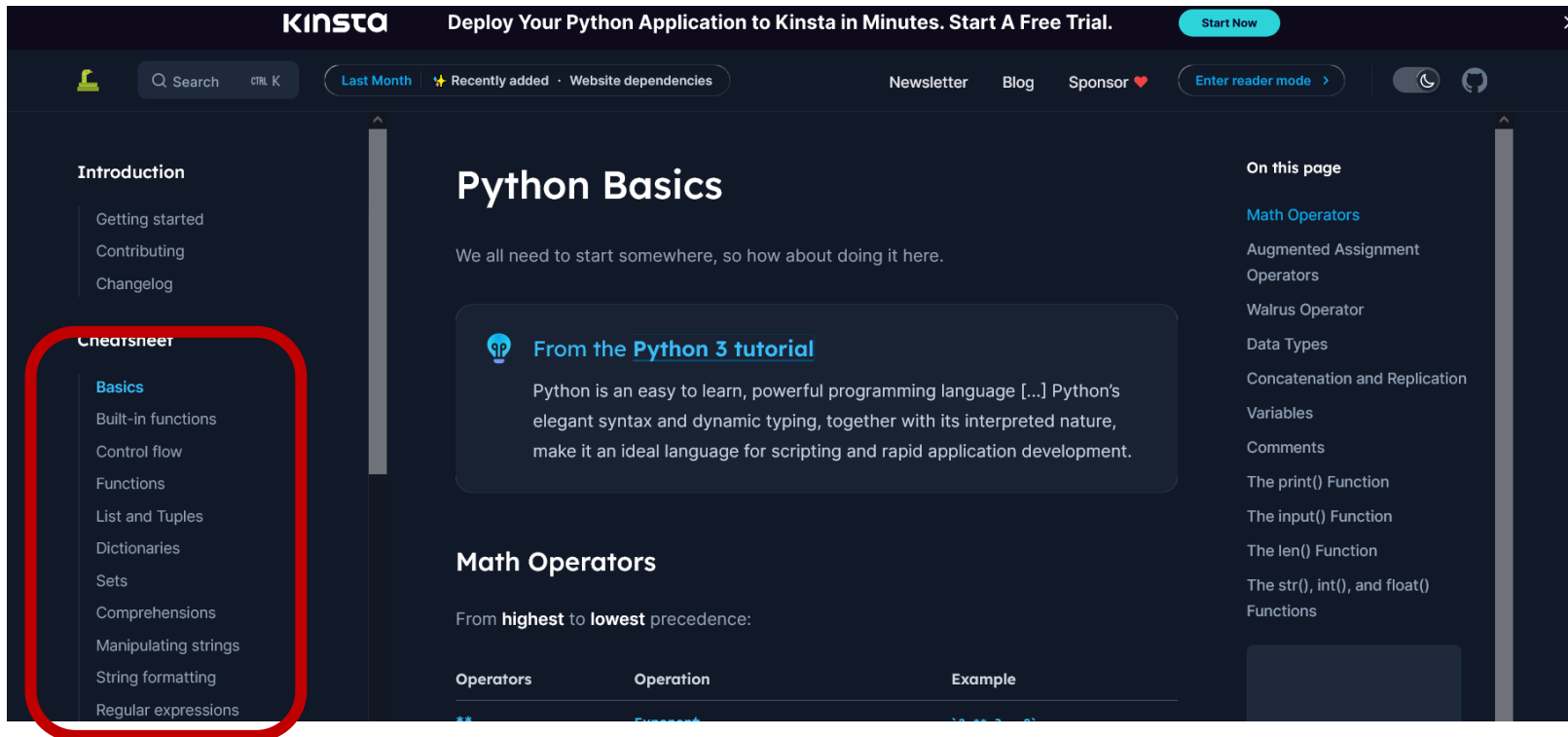
TypeError: can only concatenate str (not "int") to str
```



# Coding: elements...

- Python cheat sheet:

<https://www.pythoncheatsheet.org/cheatsheet/basics>



The screenshot shows the Python Cheatsheet website. The sidebar on the left contains a 'Cheatsheet' section with a list of topics. The 'Basics' topic is highlighted with a red rounded rectangle. The main content area is titled 'Python Basics' and includes a quote from the Python 3 tutorial. Below this, there is a section for 'Math Operators' with a table of operators and their precedence.

**Introduction**

- Getting started
- Contributing
- Changelog

**Cheatsheet**

- Basics**
- Built-in functions
- Control flow
- Functions
- List and Tuples
- Dictionaries
- Sets
- Comprehensions
- Manipulating strings
- String formatting
- Regular expressions

**Python Basics**

We all need to start somewhere, so how about doing it here.

**From the Python 3 tutorial**

Python is an easy to learn, powerful programming language [...] Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development.

**Math Operators**

From **highest** to **lowest** precedence:

Operators	Operation	Example
++	Increment	1 + 1 = 2

**On this page**

- Math Operators**
- Augmented Assignment
- Operators
- Walrus Operator
- Data Types
- Concatenation and Replication
- Variables
- Comments
- The print() Function
- The input() Function
- The len() Function
- The str(), int(), and float() Functions

# Coding: elements...

- Python cheat sheet:  
<https://www.pythoncheatsheet.org/cheatsheet/basics>

## Math Operators

From **highest** to **lowest** precedence:

Operators	Operation	Example
<code>**</code>	Exponent	<code>`2 ** 3 = 8`</code>
<code>%</code>	Modulus/Remainder	<code>`22 % 8 = 6`</code>
<code>//</code>	Integer division	<code>`22 // 8 = 2`</code>
<code>/</code>	Division	<code>`22 / 8 = 2.75`</code>
<code>*</code>	Multiplication	<code>`3 * 3 = 9`</code>
<code>-</code>	Subtraction	<code>`5 - 2 = 3`</code>
<code>+</code>	Addition	<code>`2 + 2 = 4`</code>

# Coding: elements...

- Python cheat sheet:  
<https://www.pythoncheatsheet.org/cheatsheet/built-in-functions>

The image is a screenshot of a website titled "Python built-in Functions". The website has a dark theme. At the top, there are links for "Recently added", "Website dependencies", "Newsletter", "Blog", and "Sponsor" with a heart icon. Below the header is a table with two columns: "Function" and "Description". The table lists various built-in Python functions and their descriptions. The functions listed are: abs(), aiter(), all(), any(), ascii(), bin(), bool(), breakpoint(), bytearray(), bytes(), and callable().

Function	Description
<code>abs()</code>	Return the absolute value of a number.
<code>aiter()</code>	Return an asynchronous iterator for an asynchronous iterable.
<code>all()</code>	Return True if all elements of the iterable are true.
<code>any()</code>	Return True if any element of the iterable is true.
<code>ascii()</code>	Return a string with a printable representation of an object.
<code>bin()</code>	Convert an integer number to a binary string.
<code>bool()</code>	Return a Boolean value.
<code>breakpoint()</code>	Drops you into the debugger at the call site.
<code>bytearray()</code>	Return a new array of bytes.
<code>bytes()</code>	Return a new "bytes" object.
<code>callable()</code>	Return True if the object argument is callable, False if not.

etc

# Coding: elements...

- Python cheat sheet:

<https://www.pythoncheatsheet.org/cheatsheet/control-flow>

## Comparison Operators

Operator	Meaning
<code>`==`</code>	Equal to
<code>`!=`</code>	Not equal to
<code>`&lt;`</code>	Less than
<code>`&gt;`</code>	Greater Than
<code>`&lt;=`</code>	Less than or Equal to
<code>`&gt;=`</code>	Greater than or Equal to

## Boolean Operators

There are three Boolean operators: ``and``, ``or``, and ``not``. In the order of precedence, highest to lowest they are ``not``, ``and`` and ``or``.

etc



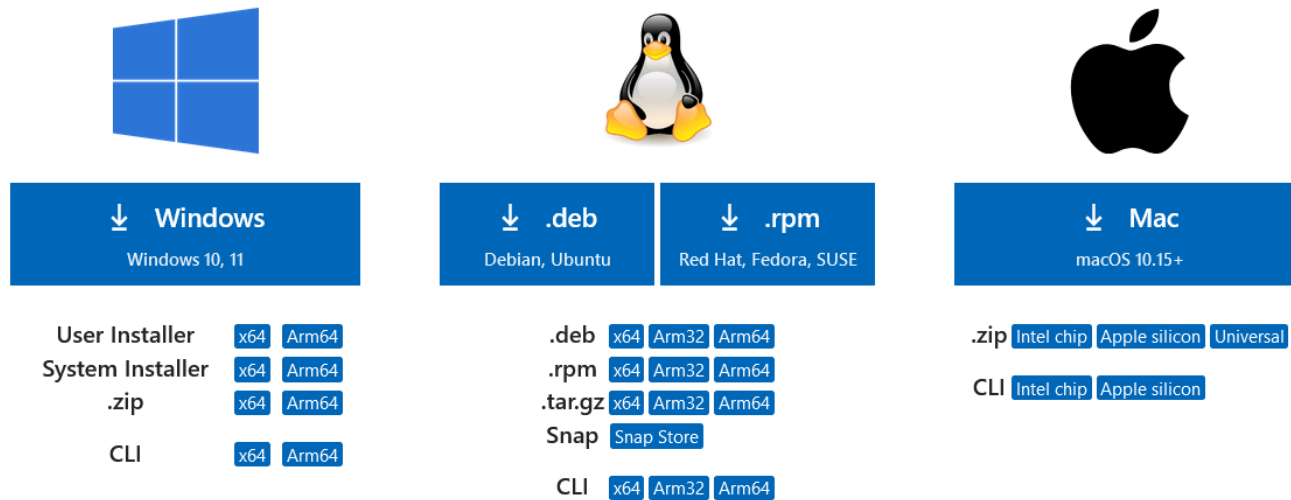
# Coding: elements...

- Python cheat sheet:
  - <https://www.pythoncheatsheet.org/cheatsheet/functions>
  - <https://www.pythoncheatsheet.org/cheatsheet/lists-and-tuples>
  - <https://www.pythoncheatsheet.org/cheatsheet/dictionaries>
  - <https://www.pythoncheatsheet.org/cheatsheet/sets>
  - <https://www.pythoncheatsheet.org/cheatsheet/comprehensions>
  - <https://www.pythoncheatsheet.org/cheatsheet/string-formatting>

# In the beginning... There was an IDE

- VS Code

<https://code.visualstudio.com/#alt-downloads>



The screenshot displays the download options for Visual Studio Code, organized into three main sections: Windows, Linux, and Mac. Each section features a platform icon at the top, a download button with a dropdown arrow, and a list of available installers with their supported architectures.

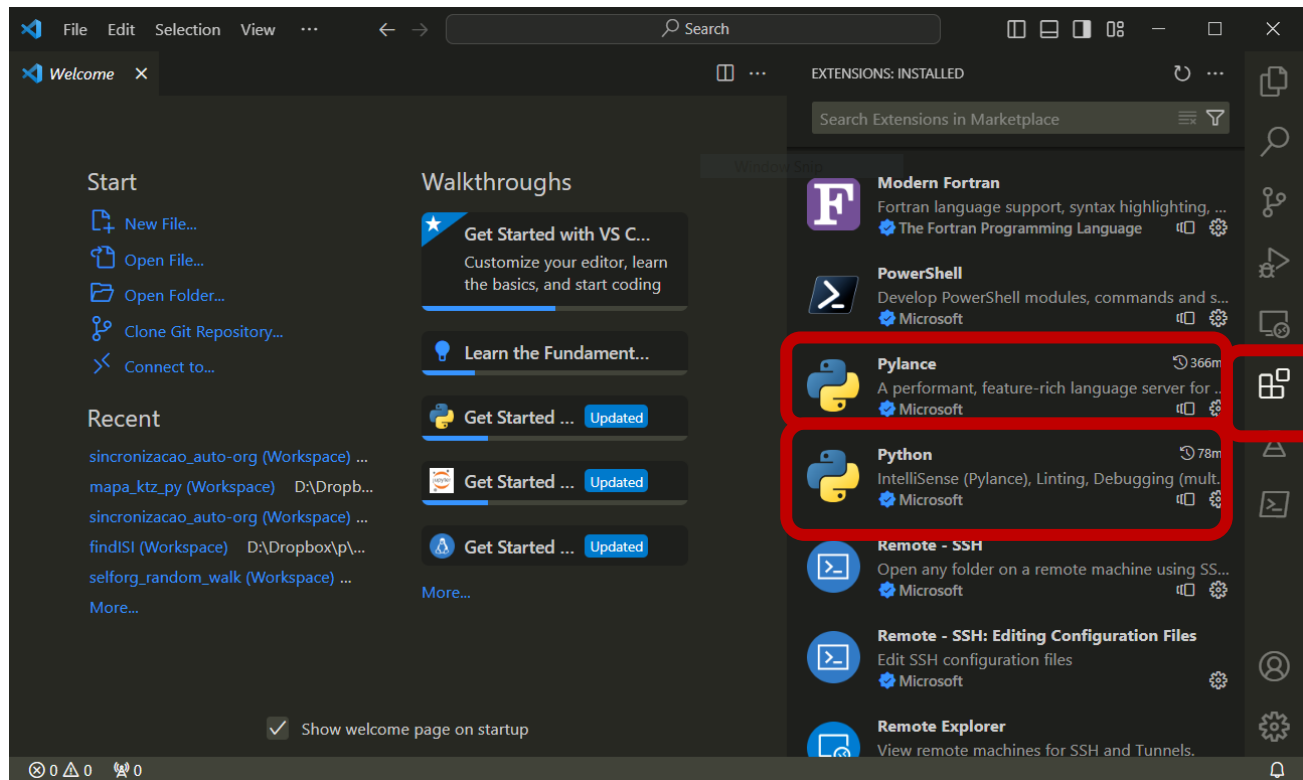
Platform	Download Button	Available Installers
Windows	Windows 10, 11	User Installer (x64, Arm64), System Installer (x64, Arm64), .zip (x64, Arm64), CLI (x64, Arm64)
Linux	.deb (Debian, Ubuntu) and .rpm (Red Hat, Fedora, SUSE)	.deb (x64, Arm32, Arm64), .rpm (x64, Arm32, Arm64), .tar.gz (x64, Arm32, Arm64), Snap (Snap Store), CLI (x64, Arm32, Arm64)
Mac	Mac (macOS 10.15+)	.zip (Intel chip, Apple silicon, Universal), CLI (Intel chip, Apple silicon)

Lightweight, Intellisense (suggestions/corrections of code snippets), syntax highlighting, Jupyter integrated support (view variable workspace on the fly)

# In the beginning... There was an IDE

- Python in VS Code:

<https://code.visualstudio.com/docs/languages/python>

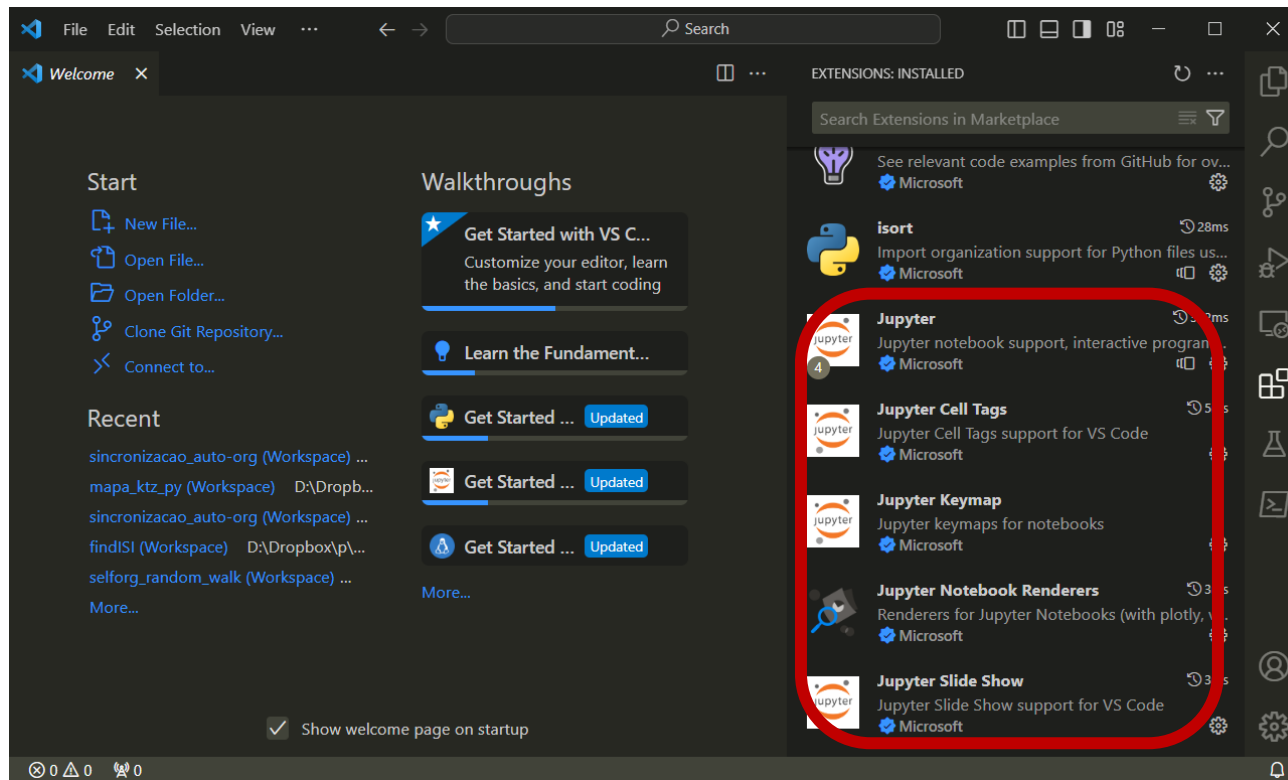


- Python
- Pylance

# In the beginning... There was an IDE

- Python in VS Code:

<https://code.visualstudio.com/docs/languages/python>



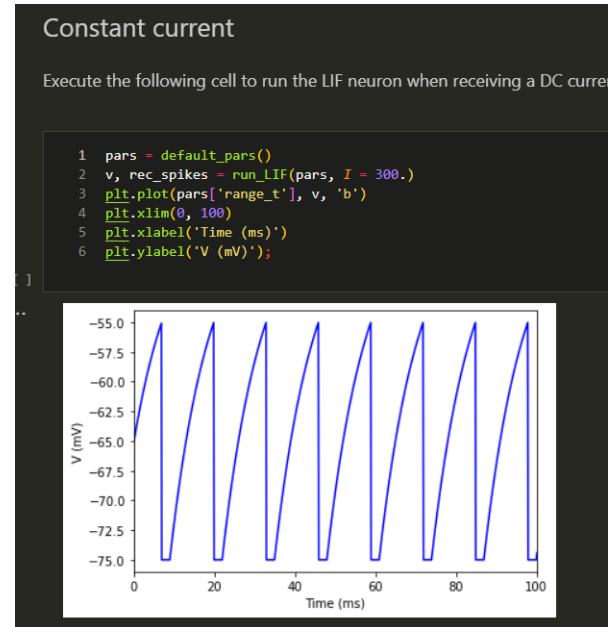
- Python
- Pylance
- Jupyter
- Other Jupyter related

# Python: must know

- Python is not a “go-to” language for performance!
- Python is optimized for user-friendliness

- **Jupyter -> literally a “notebook”**

- Code executed by parts
- You can mix code with annotations
- Direct visual I/O feedback of your code
- Good for plotting
- Good for probing unknown problems
- Usually worse performance
- Runs in the browser or in VS Code



- **Scripting -> more like an “old school program”**

- Write all the code -> run all at once
- Another script needed to analyze the data
- Better when you need performance
- Run from the terminal: `python myscript.py`
- Useful tools for performance gain:  
**numba, pythran, cython, f2py**

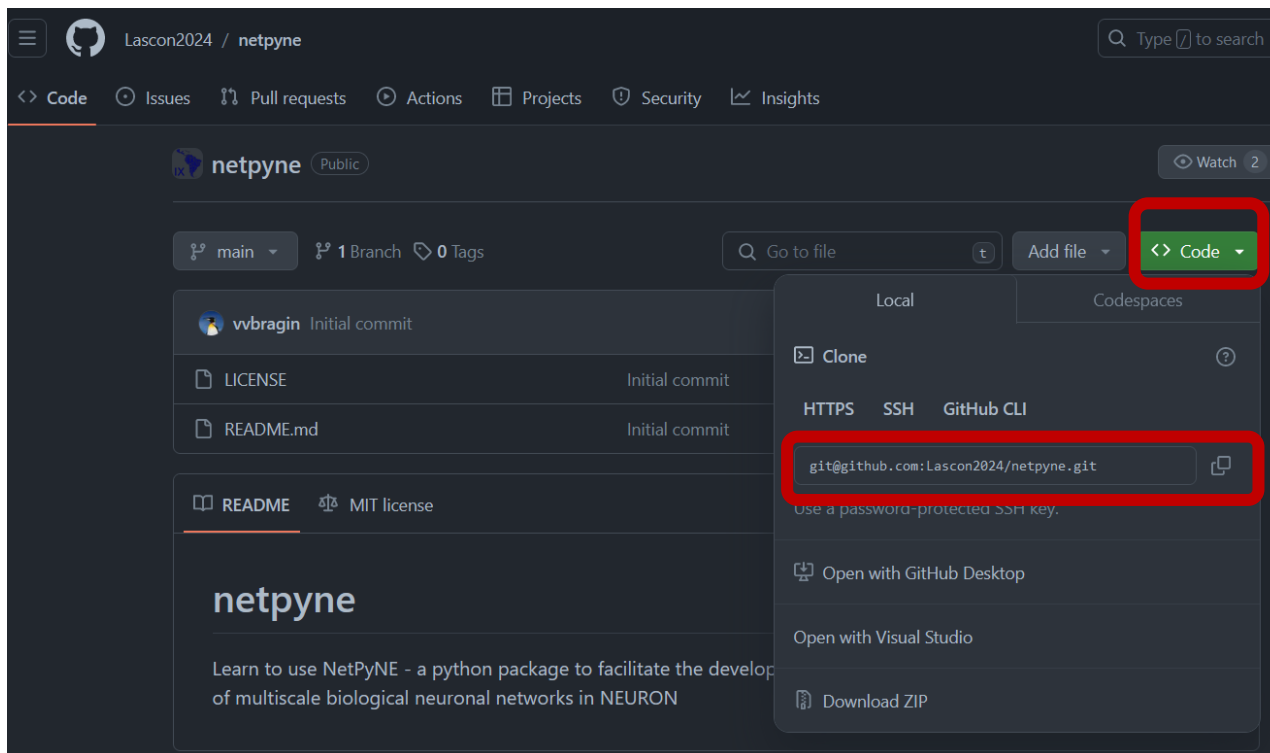
```
11 import quantities
12 import modules.output as output
13 import modules.neurons as neu
14 import modules.plot as p
15 import modules.input as inp
16 import modules.misc as misc
17
18 parser = argparse.ArgumentParser(description='Run c
19 parser = inp.add_neuron_parameters(parser)
20 #parser = inp.add_neuronparamset_parameters(parser)
21 #parser = inp.add_stimulus_parameters(parser, tStim=
22 parser = inp.add_stimulus_parameters(parser, tStim=
23 #parser = inp.add_stimulus_parameters(parser, tStim=
24 parser = inp.add_output_parameters(parser, out=['cur
25 parser = inp.add_simulation_parameters(parser, T=[50
26 parser.add_argument('-show', required=False, action
27 parser.add_argument('-savefig', required=False, act
28 parser.add_argument('-savevoltage', required=False,
29
30 args = parser.parse_args()
31 saveData = args.save
32 saveVoltage = args.savevoltage
33 saveFig = args.savefig
34 showFig = args.show
```

# Python: must know

- git <https://github.com/Lascon2024>

cd ~/my/lascon2024/netpyne

git clone [git@github.com:Lascon2024/netpyne.git](https://github.com/Lascon2024/netpyne.git) .



# Python: must know

- **numpy**: MATLAB-like vector/matrix manipulation  
<https://numpy.org/doc/stable/user/quickstart.html>

```
>>> a = np.array(1, 2, 3, 4)    # WRONG
```

```
Traceback (most recent call last):
```

```
...
```

```
TypeError: array() takes from 1 to 2 positional arguments but 4 were given
```

```
>>> a = np.array([1, 2, 3, 4]) # RIGHT
```

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<class 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<class 'numpy.ndarray'>
```



# Python: must know

- **scipy**: Scientific/math codes (based on numpy)  
<https://docs.scipy.org/doc/scipy/tutorial/index.html>
  - Interpolation (fill missing data gaps): **scipy.interpolate**
  - Statistics: **scipy.stats** (T-test, chi-squared, etc)
  - Signal processing: **scipy.signal** (Filters, find peaks, spectra)
  - Optimization: **scipy.optimize** (curve fitting, parameter estimation)
  - Integration: **scipy.integrate** (Simpson, trapezoid, etc)
  - Sparse matrices: **scipy.sparse** (large data arrays with mostly zeros)
  - I/O: **scipy.io** (savemat, loadmat -> integration with MATLAB; easy data storage – better than numpy's, and more widely compatible)

## SciPy User Guide

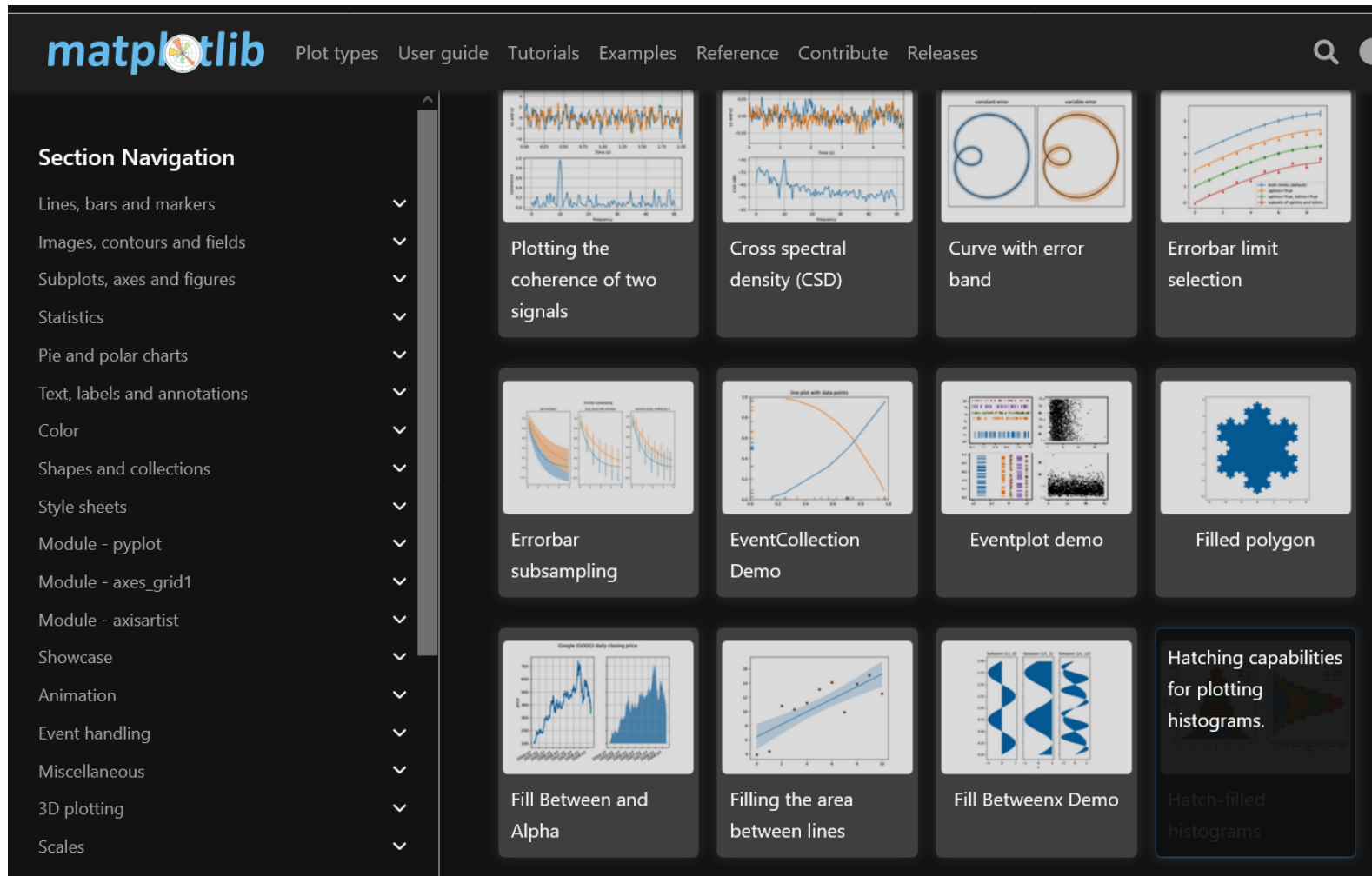
SciPy is a collection of mathematical algorithms and convenience functions built on NumPy . It adds significant power to Python by providing the user with high-level commands and classes for manipulating and visualizing data.

## Subpackages

SciPy is organized into subpackages covering different scientific computing domains. These are summarized in the following table:

# Python: must know

- **Matplotlib:** MATLAB-like figure plotting capabilities  
<https://matplotlib.org/stable/gallery/index.html>



# Python: must know

- **pandas**: data science toolkit...  
CSV and Excel file I/O (main features to me :P )  
[https://pandas.pydata.org/docs/user\\_guide/10min.html](https://pandas.pydata.org/docs/user_guide/10min.html)

## pandas.read\_excel

```
pandas.read_excel(io, sheet_name=0, *, header=0, names=None, index_col=None,
usecols=None, dtype=None, engine=None, converters=None, true_values=None,
false_values=None, skiprows=None, nrows=None, na_values=None,
keep_default_na=True, na_filter=True, verbose=False, parse_dates=False,
date_parser=_NoDefault.no_default, date_format=None, thousands=None, decimal='.',
comment=None, skipfooter=0, storage_options=None,
dtype_backend=_NoDefault.no_default, engine_kwargs=None)
```

[\[source\]](#)

# Further learning...

- Our git repo:  
<https://github.com/Lascon2024/python-tutorial>
- Scientific python lectures (Scipy, numpy examples)  
<https://github.com/jrjohansson/scientific-python-lectures/tree/master>
- Beginner concepts:  
The way of the program:  
<https://greenteapress.com/thinkpython2/html/thinkpython2002.html>  
Variables, expressions, statements:  
<https://greenteapress.com/thinkpython2/html/thinkpython2003.html>
- Intro to python:  
[https://cs111.wellesley.edu/lectures/lec\\_python\\_intro/](https://cs111.wellesley.edu/lectures/lec_python_intro/)

# Example...

Leaky IF neuron:

$$\tau \frac{dV}{dt} = V_b - V + RI$$

$$\text{If } V = \theta$$

$$V(t) = 60 \text{ mV}$$

$$V(t + dt) = V_0$$

- Determine the problem
- Identify parameters
- Identify variables

Quadratic IF neuron:

$$C \frac{dV}{dt} = G(V - V_r)(V - V_c) + I$$

$$\text{If } V = \theta$$

$$V(t) = 60 \text{ mV}$$

$$V(t + dt) = V_0$$