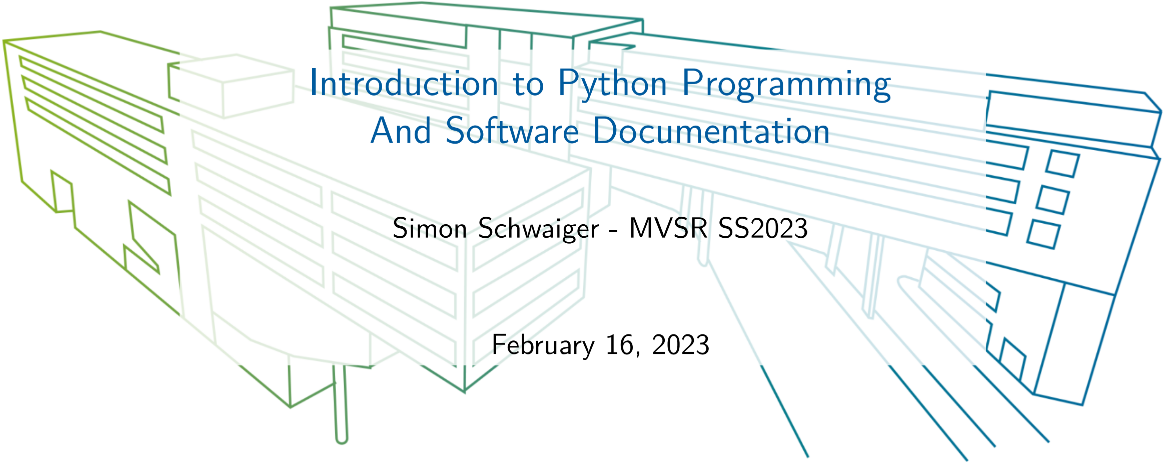


Introduction to Python Programming And Software Documentation

Simon Schwaiger - MVSR SS2023

February 16, 2023

Content

- Programming in Python
- Software Documentation

Programming in Python

Programming in Python*

Advantages

- Wide adoption
- Fast to implement
- Short code, Easy to learn
- Automatic setting of data type
- Garbage collection

Disdvantages

- Slow* runtime
- Versioning
- Packaging

Applications[1]

- Scientific computing & data science
- Web & GUI development
- Software development
- System administration

Distinction from other Languages

- No semi-column and brackets
  → Line endings and indentations
- Compiled to bytecode instructions
- C/C++ libraries provide speed

[1] *Source*: `https://www.python.org/`

Programming in Python*

Important Data Structures

- Bytes → Everything can be decoded
- Lists → Array with numbered indices
- Dicts → Array with indices of other data type

Important Modules

- Rospy → ROS implementation
- NumPy → Arrays, mathematical functions
- Scikit-Learn → Data processing algorithms

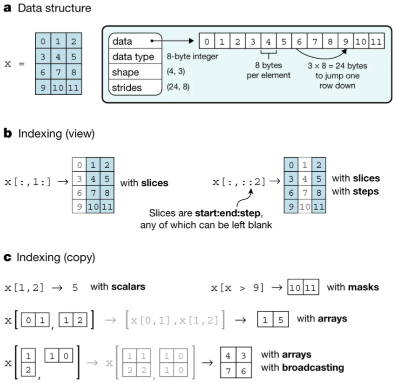


Figure: NumPy Arrays for Data Management[1]

[1] *Source*: `https://www.nature.com/articles/s41586-020-2649-2`

Programming in Python - Essential Code Syntax*

Indentations and Line Endings

- Code grouped using indentations other languages: for readability only
  → Indentations instead of brackets in C/C++
- End-of-line character indicates the end of an instruction
  → Newline instead of semi columns in C/C++

Library Imports

- *import <package name>*

Comments

- Single line comments: # *commented text*
- Multi line comments: *""" commented text """*

Programming in Python - Essential Code Syntax*

Example: Branches

```
# Checks if value has been changed from 2
if value == 2:
    print("Value␣has␣not␣been␣changed!")
elif value == 3:
    print("Value␣is␣three!")
else:
    print("Value␣has␣been␣changed!")
```

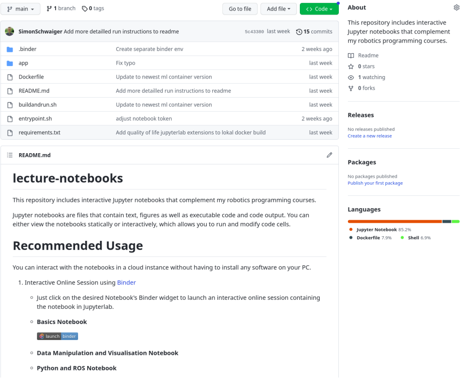
Example: Loops

```
# Prints out numbers from 0 to 4
for i in range(5):
    print(i)
```

Programming in Python - Essential Code Syntax*

Python Programming Resources

- Essential Syntax Notebook[1]
- W3Schools Python Tutorial[2]
- Official Python Documentation[3]



SimonSchwaiger Add more detailled run instructions to readme 5c43380 last week 15 commits

.binder	Create separate binder env	2 weeks ago
app	Fix typo	last week
Dockerfile	Update to newest ml container version	last week
README.md	Add more detailled run instructions to readme	last week
buildandrun.sh	Update to newest ml container version	last week
entrypoint.sh	adjust notebook token	2 weeks ago
requirements.txt	Add quality of life jupyterlab extensions to lokal docker build	last week

About

This repository includes interactive Jupyter notebooks that complement my robotics programming courses.

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

- Jupyter Notebook 85.2%
- Dockerfile 7.9%
- Shell 6.9%

README.md

lecture-notebooks

This repository includes interactive Jupyter notebooks that complement my robotics programming courses.

Jupyter notebooks are files that contain text, figures as well as executable code and code output. You can either view the notebooks statically or interactively, which allows you to run and modify code cells.

Recommended Usage

You can interact with the notebooks in a cloud instance without having to install any software on your PC.

1. Interactive Online Session using Binder
   - Just click on the desired Notebook's Binder widget to launch an interactive online session containing the notebook in Jupyterlab.
   - **Basics Notebook**
   - **Data Manipulation and Visualisation Notebook**
   - **Python and ROS Notebook**

Figure: The linked repository contains notebooks that explain essential code syntax[1]

[1] *See*: `https://github.com/SimonSchwaiger/lecture-notebooks`

[2] *See*: `https://www.w3schools.com/python/`

[3] *See*: `https://docs.python.org/3/`

Programming in Python

Debugging Tools (Optional)

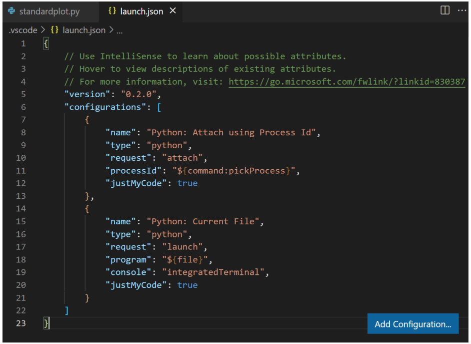
- Python extension for Visual Studio Code

→ Break-points and stepping through code

Assertions (Highly Recommended)

- Condition → if not met, program is terminated
- Format: assert *condition*, *error message*
- Example:

```
assert x >= 6, "x is smaller than 6!"
```



```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python: Attach using Process Id",
            "type": "python",
            "request": "attach",
            "processId": "${command:pickProcess}",
            "justMyCode": true
        },
        {
            "name": "Python: Current File",
            "type": "python",
            "request": "launch",
            "program": "${file}",
            "console": "integratedTerminal",
            "justMyCode": true
        }
    ]
}
```

Figure: Example Configuration of VS Code's Python Debugger[1]

[1] *Source*: https://code.visualstudio.com/docs/python/debugging

Programming in Python*

Classes

→ Object, consisting of member variables and methods (i.e. data and logic)

- **Member variables** contain an initial value and are modifiable
- **Methods** contain functions that process/modify class members and external variables

Objects vs. Data Structures

- Objects abstract
- Objects contain functionality
- Data structures only contain data.

→ Find balance between abstraction and flat data representation

Programming in Python

Iterators in Python

→ Iteration in Python is slow! → Use matrix/vector operations for performance

- **Iteration over list**

  `for i in range(3)` or `for i in ["item1", "item2"]`
- **Numbered iteration over list**

  ```
  for index, item in enumerate(["hello", "world"])
  ```
- **Simultaneous iteration over two lists**

  ```
  for item1, item2 in zip(list1, list2)
  ```
- **List Comprehension**

  ```
  [ item for item in list if condition==True ]
  ```

Programming in Python - List Comprehension Examples

- **List containing default parameters**

```
placeholder = [ -1 for _ in range(numJoints) ]
```

- **Decoding of string containing jointstate feedback**

```
jointstate = [ float(entry) for entry in fbString.split(", ") ]
```

- **Check if each value is in bounds**

```
if False in [
    lower <= value <= higher
    for lower, higher, value in zip(JsMinPos, JsMaxPos, JsPos)
]:
    reward = -1
```

Programming in Python

Inheritance

- Classes can inherit from each other
- Members and functions are adopted
- Child class *contains* inherited class

Interfaces

- Definition of method calls
- Definition of required member variables
- Class adopts interface using inheritance
- Special handling in Python

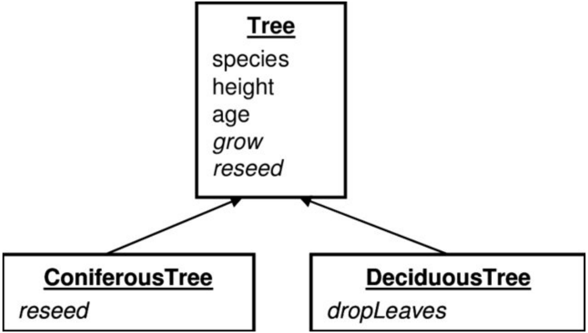
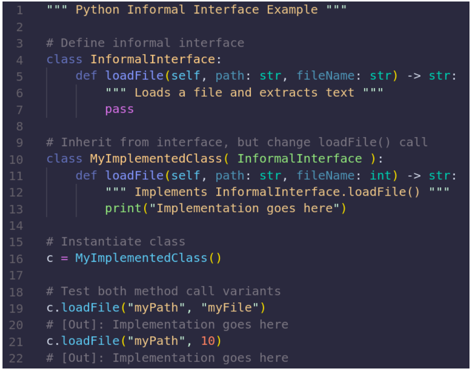


Figure: Principle of Inheritance explained with Trees[1]

[1] *Source*: `https://www.researchgate.net/publication/228839172_Meeting_the_Challenge_of_Complexity/figures?lo=1`

Programming in Python



```python
""" Python Informal Interface Example """

# Define informal interface
class InformalInterface:
    def loadFile(self, path: str, fileName: str) -> str:
        """ Loads a file and extracts text """
        pass

# Inherit from interface, but change loadFile() call
class MyImplementedClass( InformalInterface ):
    def loadFile(self, path: str, fileName: int) -> str:
        """ Implements InformalInterface.loadFile() """
        print("Implementation goes here")

# Instantiate class
c = MyImplementedClass()

# Test both method call variants
c.loadFile("myPath", "myFile")
# [Out]: Implementation goes here
c.loadFile("myPath", 10)
# [Out]: Implementation goes here
```

Figure: Interfaces are not available in Python due to the abstract typing system. Therefore, this example with an implementation changing the interface, will not produce a runtime error[1]

[1] *Based on*: `https://realpython.com/python-interface/`

Programming in Python

Type Checking

→ Sometimes, dynamic type system not wanted

e.g. due to debugging or for code analysis

Static Code Analysis (Optional)

- Analyse code to find faults
- Annotations based on Python Enhancement Proposals (PEP)[1]
- Describe variable types and function arguments

```
def thisIsAFunction(inputInt: int, inputString: str) -> np.ndarray:
```

- Type checker matches annotations with writes to a variable

[1] *See*: https://docs.python.org/3/library/typing.html

Programming in Python

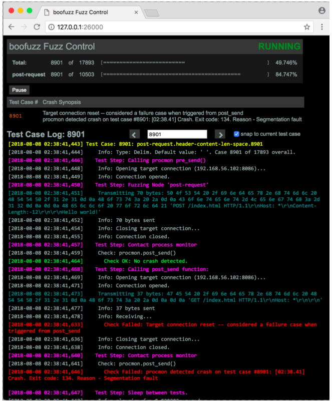
Dynamic Code Analysis (Out of Scope for Project)

- Execution of (parts of) a program
- Test cases ↔ Randomised input patterns
- Check if expected output achieved

Code Coverage = Fraction of program code that was executed/tested during analysis

Fuzzing

- Application of pseudo-randomised input patterns
- **Goal**: Achieve unexpected system behaviour



boofuzz Fuzz Control RUNNING

```
Total:        8901  of  17893  [========================                  ]  49.746%
post-request  8901  of  10503  [========================================  ]  84.747%
```

Test Case #	Crash Synopsis
8901	Target connection reset -- considered a failure case when triggered from post_send procmon detected crash on test case #8901: [02:38.41] Crash. Exit code: 134. Reason - Segmentation fault

Test Case Log: 8901

```
[2018-08-08 02:38:41,443] Test Case: 8901: post-request.header-content-len-space.8901
[2018-08-08 02:38:41,444]     Info: Type: Delim. Default value: ' '. Case 8901 of 17893 overall.
[2018-08-08 02:38:41,446]    Test Step: Calling procmon pre_send()
[2018-08-08 02:38:41,448]     Info: Opening target connection (192.168.56.102:8086)...
[2018-08-08 02:38:41,449]     Info: Connection opened.
[2018-08-08 02:38:41,450]    Test Step: Fuzzing Node 'post-request'
[2018-08-08 02:38:41,451]     Transmitting 70 bytes: 50 4f 53 54 20 2f 69 6e 64 65 78 2e 68 74 6d 6c 20
48 54 54 50 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 2a 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 68 3a 2d
31 32 0d 0a 0d 0a 48 65 6c 6c 6f 20 77 6f 72 6c 64 21 'POST /index.html HTTP/1.1\r\nHost: *\r\nContent-
Length:-12\r\n\r\nHello world!'
[2018-08-08 02:38:41,452]     Info: 70 bytes sent
[2018-08-08 02:38:41,454]     Info: Closing target connection...
[2018-08-08 02:38:41,455]     Info: Connection closed.
[2018-08-08 02:38:41,457]    Test Step: Contact process monitor
[2018-08-08 02:38:41,459]     Check: procmon.post_send()
[2018-08-08 02:38:41,464]       Check OK: No crash detected.
[2018-08-08 02:38:41,468]    Test Step: Calling post_send function:
[2018-08-08 02:38:41,469]     Info: Opening target connection (192.168.56.102:8086)...
[2018-08-08 02:38:41,471]     Info: Connection opened.
[2018-08-08 02:38:41,473]     Transmitting 37 bytes: 47 45 54 20 2f 69 6e 64 65 78 2e 68 74 6d 6c 20 48
54 54 50 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 2a 0d 0a 0d 0a 'GET /index.html HTTP/1.1\r\nHost: *\r\n\r\n'
[2018-08-08 02:38:41,477]     Info: 37 bytes sent
[2018-08-08 02:38:41,478]     Info: Receiving...
[2018-08-08 02:38:41,633]       Check Failed: Target connection reset -- considered a failure case when
triggered from post_send
[2018-08-08 02:38:41,636]     Info: Closing target connection...
[2018-08-08 02:38:41,638]     Info: Connection closed.
[2018-08-08 02:38:41,640]    Test Step: Contact process monitor
[2018-08-08 02:38:41,641]     Check: procmon.post_send()
[2018-08-08 02:38:41,646]       Check Failed: procmon detected crash on test case #8901: [02:38.41]
Crash. Exit code: 134. Reason - Segmentation fault

[2018-08-08 02:38:41,647]    Test Step: Sleep between tests.
```

Figure: Fuzzing of Network Interface[1]

[1] *Source*: `https://github.com/jtpereyda/boofuzz`

Programming in Python*

Python Packaging

- Packages → "Libraries" that provide functionality
- Installation using PyPI[1], a database of Python packages

Dependency Management

- **Pip package manager** → installs packages from PyPI using *pip install <package name>*
- **Virtual environments** → contain sets of packages installed using pip
- **requirements.txt files** → list of all packages installed in a virtual environment.

  → Created using `pip freeze > requirements.txt`

  → Installed using `pip install -r requirements.txt`

[1] *See for more information*: https://packaging.python.org/en/latest/tutorials/installing-packages/#id18

Software Documentation

Software Documentation

First Step → Easily understandable Code

- Clear formatting
- Minimise exotic constructs
- Meaningful comments (code:comments → about 60%:40%)
- Follow Python coding guidelines[1]

Documentation Tools

- Autogenerated Documentation (e.g. using Doxygen or Sphinx)
- Software flowchart (recommendation: `https://app.diagrams.net/`)
- Jupyter notebook

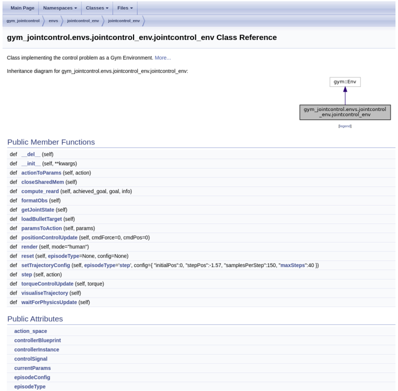
[1] *See for more Information*: `https://peps.python.org/pep-0008/`

Software Documentation

Autogenerated Documentation

- Documentation generated based on comments
- Declaration of variable types possible
- Syntax highly dependent on tool
- Result: html website

→ Python Go-To's: Sphinx, Doxygen



gym_jointcontrol.envs.jointcontrol_env.jointcontrol_env Class Reference

Class implementing the control problem as a Gym Environment. More...

Inheritance diagram for gym_jointcontrol.envs.jointcontrol_env.jointcontrol_env:

Public Member Functions

```
def  __del__ (self)
def  __init__ (self, **kwargs)
def  actionToParams (self, action)
def  closeSharedMem (self)
def  compute_reard (self, achieved_goal, goal, info)
def  formatObs (self)
def  getJointState (self)
def  loadBulletTarget (self)
def  paramsToAction (self, params)
def  positionControlUpdate (self, cmdForce=0, cmdPos=0)
def  render (self, mode="human")
def  reset (self, episodeType=None, config=None)
def  setTrajectoryConfig (self, episodeType='step', config={ "initialPos":0, "stepPos":-1.57, "samplesPerStep":150, "maxSteps":40 })
def  step (self, action)
def  torqueControlUpdate (self, torque)
def  visualiseTrajectory (self)
def  waitForPhysicsUpdate (self)
```

Public Attributes

```
action_space
controllerBlueprint
controllerInstance
controlSignal
currentParams
episodeConfig
episodeType
```

Figure: Example of automatically generated documentation using Doxygen and rosdoc-lite.

Software Documentation*

Software Flowcharts

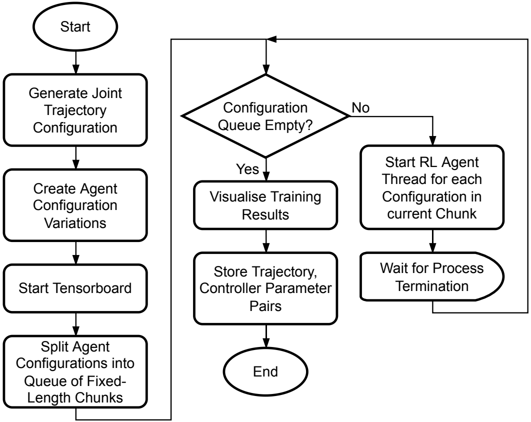

Start
Generate Joint Trajectory Configuration
Create Agent Configuration Variations
Start Tensorboard
Split Agent Configurations into Queue of Fixed-Length Chunks
Configuration Queue Empty?
No
Yes
Visualise Training Results
Store Trajectory, Controller Parameter Pairs
End
Start RL Agent Thread for each Configuration in current Chunk
Wait for Process Termination

Figure: Example Flowchart for an Application of Machine Learning in Robotics

Software Documentation*

Software Flowcharts

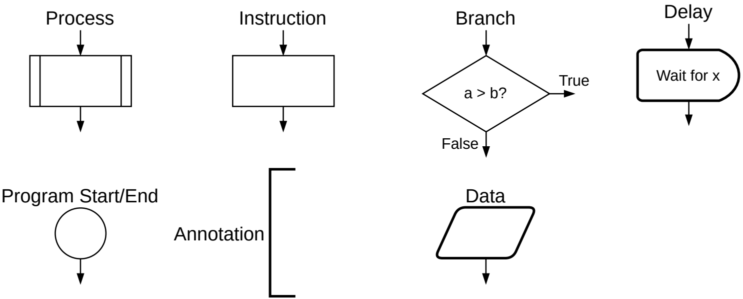


Figure: Important Flowchart Symbols

Software Documentation*

Jupyter Notebooks Consist of

- Executable Python code
- Terminal output
- Data visualisations
- Markdown/HTML documentation

JupyterLab

- Web-based development environment
- Compatible with ROS
- Visualisations using Matplotlib (`https://matplotlib.org/`)

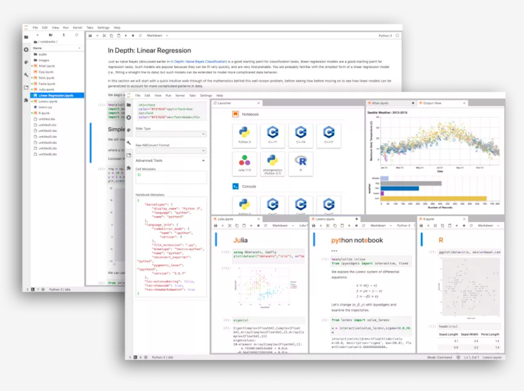


Figure: Notebooks consisting of Code, Documentation and Data Visualisations[1]

[1] *Source*: `https://jupyter.org/`

Software Documentation*

Data Visualisation in Python

Matplotlib

- Data visualisation library → similar to Matlab
- Static, animated and interactive plots
- Compatible with notebooks

Seaborn

- Statistical data visualisation using Matplotlib
- Deeply integrated with Pandas and NumPy

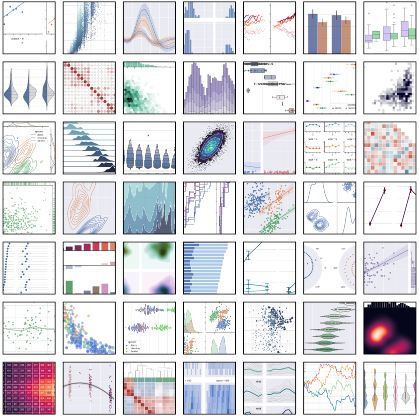


Figure: Seaborn Example Gallery[1]

[1] *Source*: `https://seaborn.pydata.org/examples/index.html`

Software Documentation - Basic Plot Types

Lineplot

- Plots connected (x, y) pairs
- Line of defined width, colour, opacity
- Curves possible using high enough sampling rate

Scatterplot

- Plots individual datapoints (x, y) on 2D plane
- Points of defined size, colour, opacity, shape

→ Can be combined into single plot

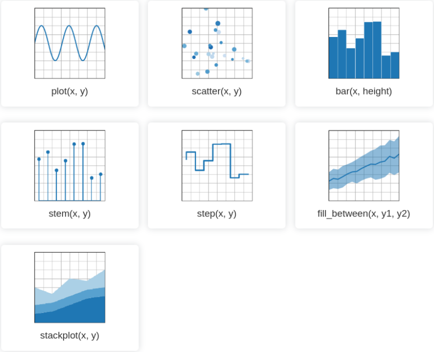


Figure: Matplotlib Example Gallery[1]

[1] *More plot types and source*: `https://matplotlib.org/stable/plot_types/index`

Software Documentation - Visualisation Examples

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(xdata1, ydata1, color='tab:blue')
ax.plot(xdata2, ydata2, color='tab:orange'

ax.set_xlim([0, 1])
ax.set_ylim([0, 1])
ax.set_title('line␣plot␣with␣data␣points')

plt.show()
```

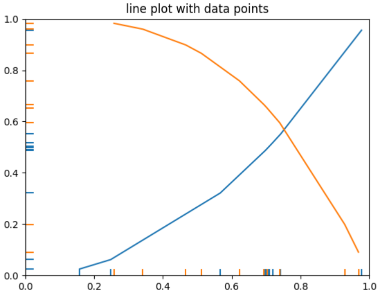


Figure: Matplotlib Lineplot Example[1]

[1] *Source*: `https://matplotlib.org/stable/gallery/lines_bars_and_markers/eventcollection_demo.html`

Software Documentation - Visualisation Examples

```
fig, ax = plt.subplots()
ax.scatter(delta1[:-1], delta1[1:],
           c=close, s=volume, alpha=0.5)

ax.set_xlabel(r'$\Delta_i$', fontsize=15)
ax.set_ylabel(r'$\Delta_{i+1}$',
              fontsize=15)
ax.set_title('Volume␣and␣percent␣change')

ax.grid(True)
fig.tight_layout()
plt.show()
```

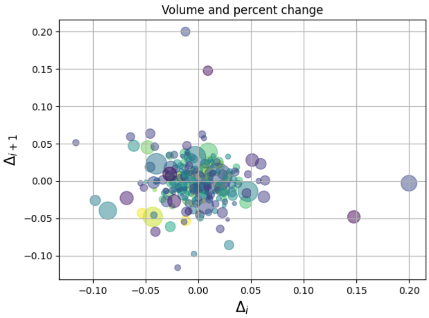


Figure: Matplotlib Scatterplot Example[1]

[1] *Source*: `https://matplotlib.org/stable/gallery/lines_bars_and_markers/scatter_demo2.html`

Software Documentation - ROS Data Visualisation*

```
ax.scatter(wallPositions[:,1],
           wallPositions[:,0],
           label="Walls")
...
for line in edgeLines:
    x0, y0 = line[0]
    x1, y1 = line[1]
    x = [x0, x1]
    y = [y0, y1]
    ax.plot(x, y, c=colourScheme["twblue"])
...
ax.grid()
ax.legend()
plt.show()
```

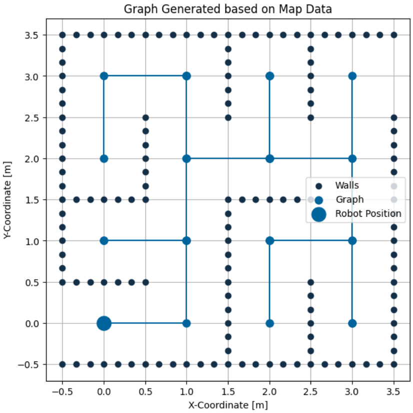


Figure: Simulated Robot, Map and Graph for Search