Linnæus University

External Packages and Project Introduction

Tobias Andersson Gidlund, Jonas Lundberg and Neda Maleki

Agenda

- External Packages
  - Famous ones
  - What is available in our Jupyter?
- Project Introduction
  - Bachelor Programme Project
  - Yatzy
  - What is needed for what grade

External Packages 📦

More Than Python

- Python is a very complete and powerful language, but there is always more that can be done
- External packages are building blocks that extend Python's functionality
- Collections of modules and other resources
- Can contain:
  - Python modules (.py files)
  - C/C++ extensions
  - Data files
  - Documentation
  - Tests

Package Development

- Languages used for developing packages:
  - Pure Python packages
    - It is Python, so easiest to use Python
  - Hybrid packages:
    - C/C++ extensions (via CPython API)
    - Cython (Python-like syntax compiled to C)
    - Rust (via PyO3)
    - Fortran (scientific computing)

Why Multiple Languages?

- Pros
  - Performance optimisation
  - Integration with system libraries
  - Legacy code compatibility
  - Hardware acceleration
- Cons
  - Development complexity
  - Build system requirements
  - Platform compatibility
  - Maintenance overhead
  - Learning curve

Notable Packages

- Scikit-learn
  - Created by: David Cournapeau (2007)
  - Purpose: Machine learning
  - Used by: Data scientists, ML engineers
  - Key features: Classification, regression, clustering
- TensorFlow
  - Created by: Google Brain team (2015)
  - Purpose: Deep learning and neural networks
  - Used by: AI researchers, ML engineers
  - Key features: Neural network modelling, GPU acceleration

Famous Python Packages

- NumPy
  - Created by: Travis Oliphant (2006)
  - Purpose: Scientific computing, array operations
  - Used by: Data scientists, researchers
  - Key features: Multi-dimensional arrays, mathematical functions
- Pandas
  - Created by: Wes McKinney at AQR Capital (2008)
  - Purpose: Data manipulation and analysis
  - Used by: Data analysts, financial sector
  - Key features: DataFrames, time series analysis

More Packages

- Django
  - Created by: Adrian Holovaty and Simon Willison (2005)
  - Purpose: Web development
  - Used by: Web developers
  - Key features: Admin interface, ORM, security
- Matplotlib
  - Created by: John Hunter (2003)
  - Purpose: Data visualisation
  - Used by: Scientists, analysts
  - Key features: Publication-quality figures, multiple plot types

In Jupyter

- Jupyter itself works with many different packages
  - The exception is GUI related frameworks (mostly)
- The Jupyter installed on LNU only supports a small number of external packages
  - This to reduce the size of the image that needs to be spun up each time
- It does, however, contain a couple of them, most notably
  - NumPy
  - Matplotlib

NumPy

- Can be found at https://numpy.org/
- Core features:
  - Multi-dimensional array objects
  - Mathematical functions
  - Broadcasting capabilities
  - Linear algebra operations
  - Random number generation

First Example, Simple List

- This will create lists in different ways using NumPy

```python
import numpy as np

simple_list = [1, 2, 3, 4, 5]
list_1 = np.array(simple_list)
print("Array from list:", list_1)

list_2 = np.zeros(5)
list_3 = np.ones(5)
list_4 = np.arange(0, 10, 2)

print('Zeros:', list_2)
print('Ones:', list_3)
print('Range with step 2:', list_4)
```

```
Array from list: [1 2 3 4 5]
Zeros: [0. 0. 0. 0. 0.]
Ones: [1. 1. 1. 1. 1.]
Range with step 2: [0 2 4 6 8]
```

Key Observations

- The dots indicate that these are floating-point numbers (like 1.0), not integers
  - `arange()` still uses integers
- Functions like `np.ones()` and `np.zeros()` default to `float64`
  - This is a special implementation for NumPy
  - Optimised for being used in lists
  - Has better memory efficiency than the normal float in Python
- Float types are default for mathematical compatibility

Basic Array (List) Operations

```python
import numpy as np

arr = np.array([1, 2, 3, 4])

# Multiple operations
print('Original array:', arr)
print('Add 2:', arr + 2)
print('Multiply by 3:', arr * 3)
print('Square:', arr ** 2)
print('Square root:', np.sqrt(arr))
```

```
Original array: [1 2 3 4]
Add 2: [3 4 5 6]
Multiply by 3: [ 3  6  9 12]
Square: [ 1  4  9 16]
Square root: [1.         1.41421356 1.73205081 2.        ]
```

Matrix Example

```python
import numpy as np

matrix = np.array([[1, 2, 3],
                   [4, 5, 6]])

print('Array shape:', matrix.shape)
print('Number of dimensions:', matrix.ndim)
print('Total elements:', matrix.size)
print('Data type:', matrix.dtype)
print('Element size in bytes:', matrix.itemsize)
```

```
Array shape: (2, 3)
Number of dimensions: 2
Total elements: 6
Data type: int64
Element size in bytes: 8
```

Matrix Operations Example

```
import numpy as np

A = np.array([[1, 2],
              [3, 4]])
B = np.array([[5, 6],
              [7, 8]])

print('Matrix A:')
print(A)
print('\nMatrix B:')
print(B)
print('\nMatrix multiplication (A @ B):')
print(A @ B)
print('\nElement-wise multiplication (A * B):')
print(A * B)
```

Output from Matrix Operations Example

```
Matrix A:
[[1 2]
 [3 4]]

Matrix B:
[[5 6]
 [7 8]]

Matrix multiplication (A @ B):
[[19 22]
 [43 50]]

Element-wise multiplication (A * B):
[[ 5 12]
 [21 32]]
```

Matplotlib

- Can be found at https://matplotlib.org/
- Created to enable Python to plot EEG data
- Needed MATLAB-like plotting capabilities
  - Named after MATLAB's plotting features
- Wanted to reduce licensing costs
- Aimed for publication-quality output

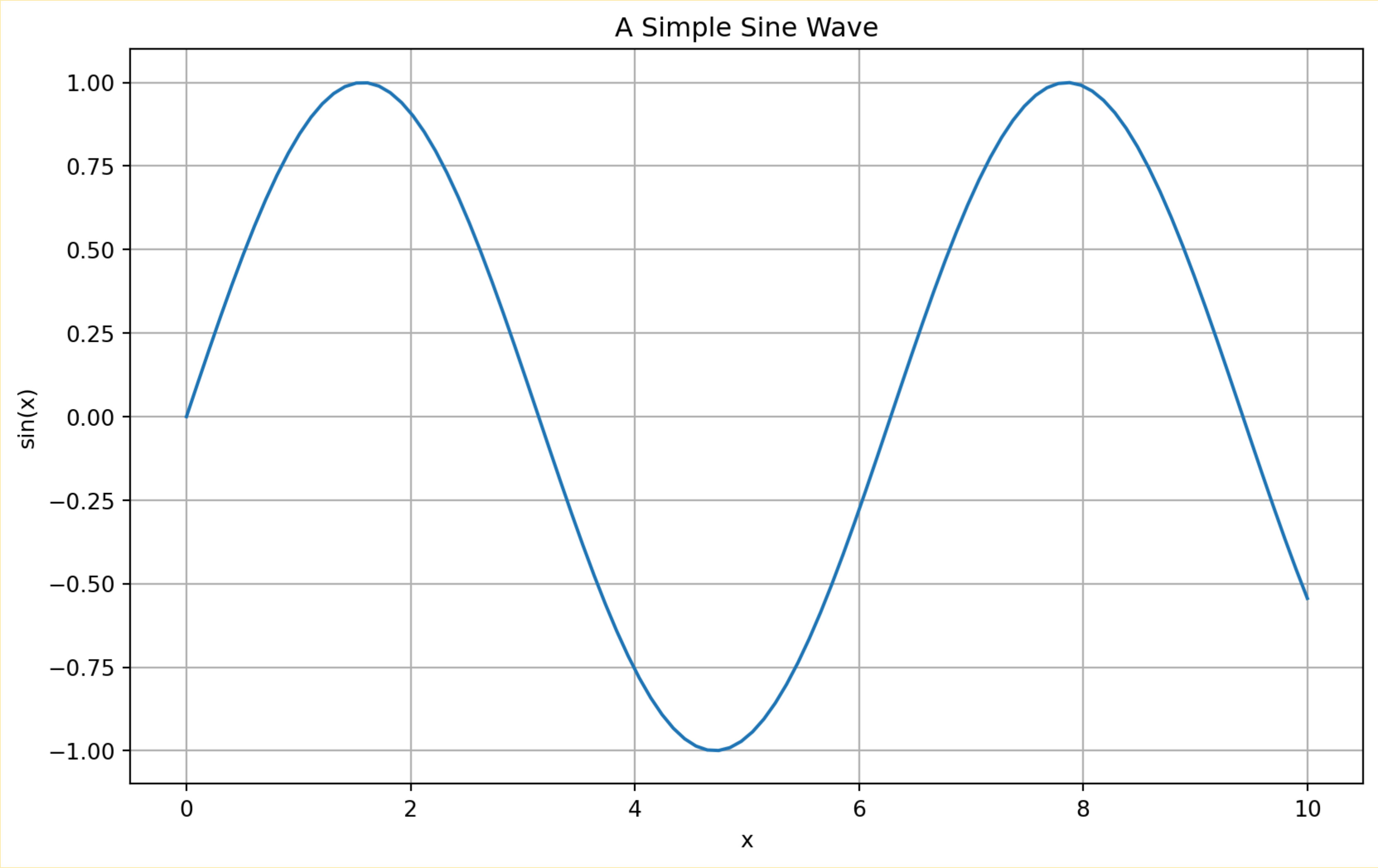
First Example

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)
y = np.sin(x)

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(x, y)
plt.title('A Simple Sine Wave')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.grid(True)
plt.show()
```

First Example Output



Explanation

- `figsize=(10, 6)`: Sets the figure size in inches
- `plot()`: Creates the line plot
- `title()`, xlabel(), ylabel(): Add labels
- `grid()`: Adds grid lines
- `show()`: Displays the plot

Several Lines

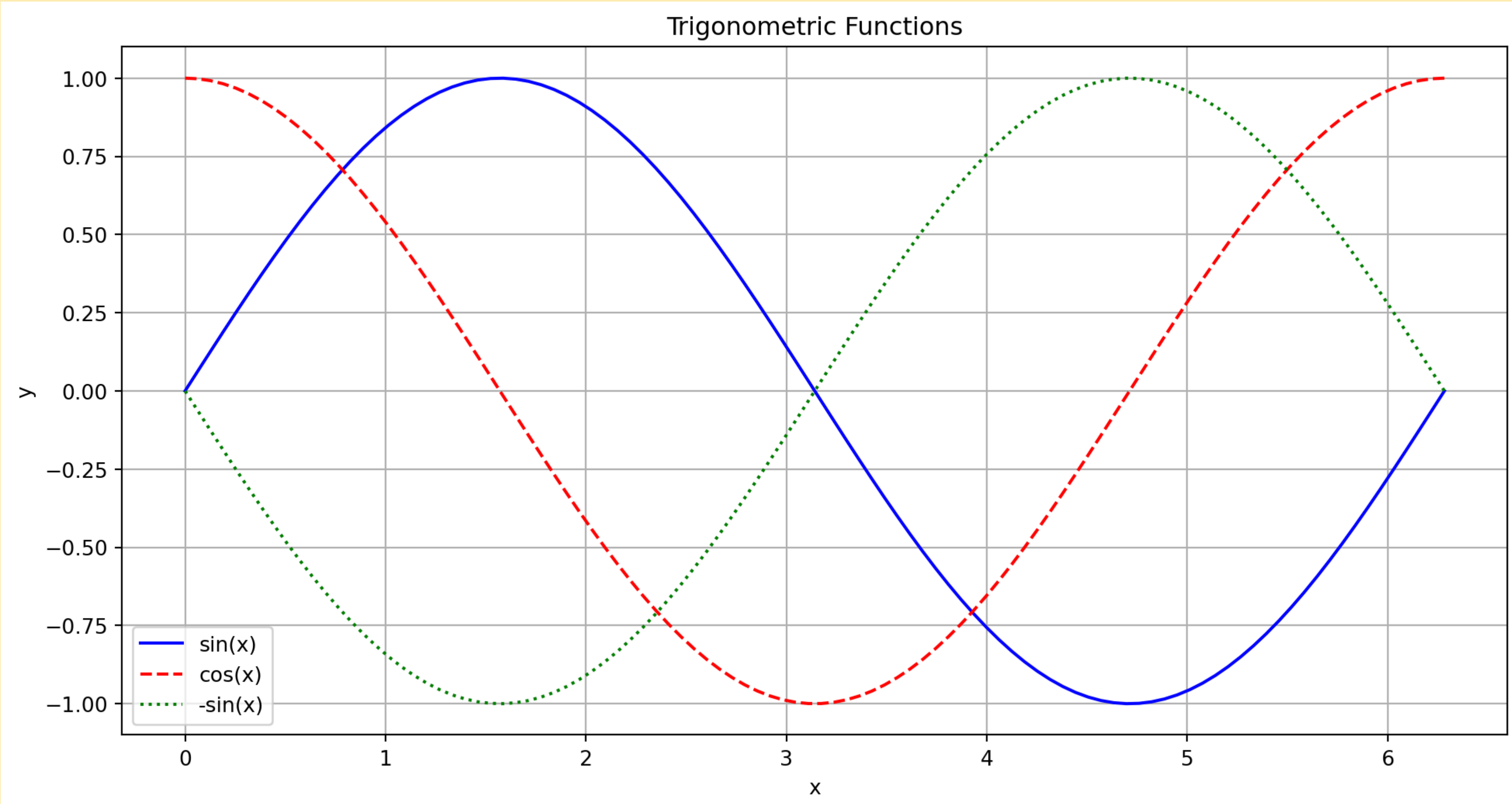
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 100)

plt.figure(figsize=(12, 6))
plt.plot(x, np.sin(x), label='sin(x)', color='blue')
plt.plot(x, np.cos(x), label='cos(x)', color='red', linestyle='--')
plt.plot(x, -np.sin(x), label='-sin(x)', color='green', linestyle=':')

plt.title('Trigonometric Functions')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()
```

Several Lines Output

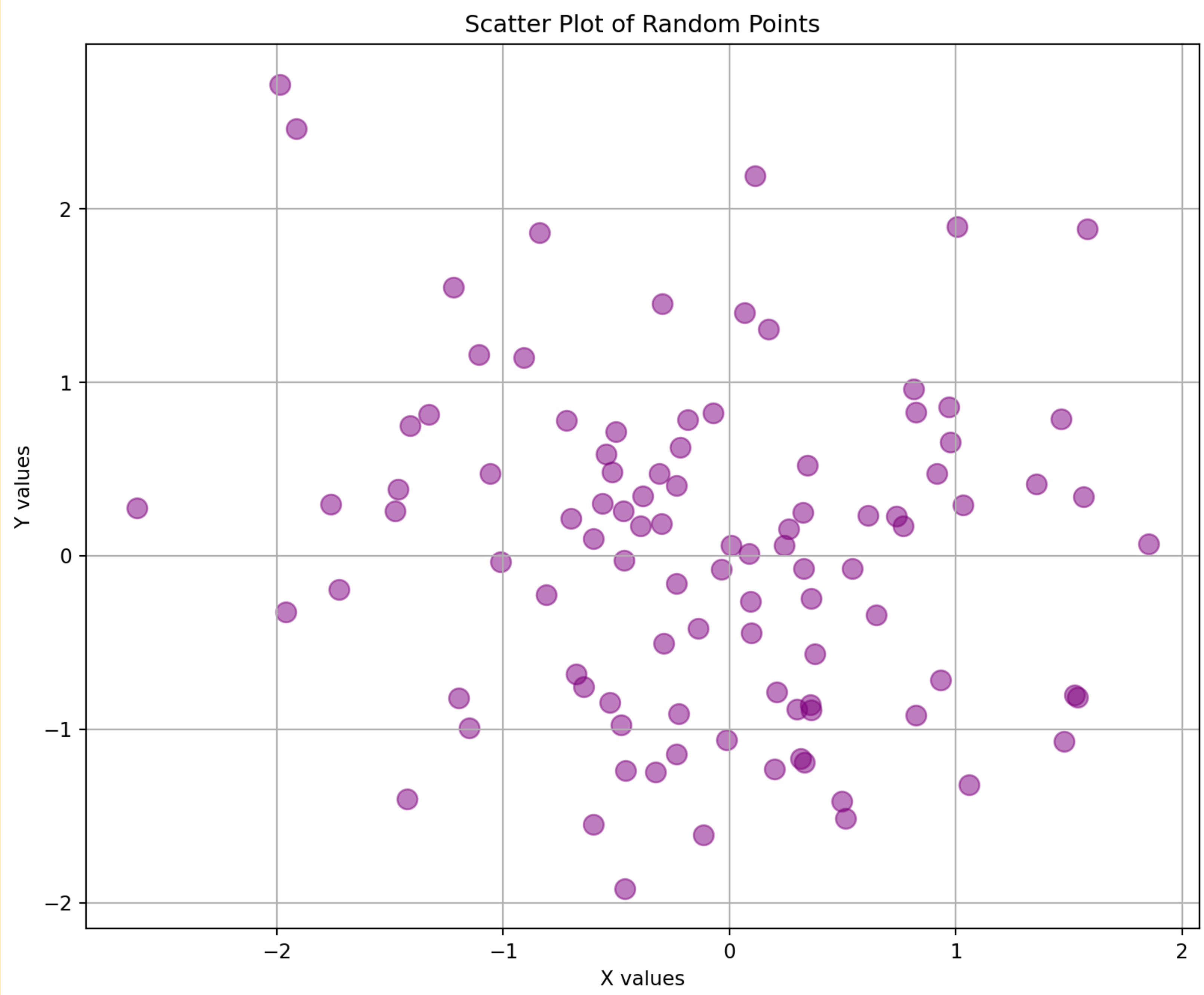


Creating Scatter Plots

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)
x = np.random.normal(0, 1, 100)
y = np.random.normal(0, 1, 100)

plt.figure(figsize=(10, 8))
plt.scatter(x, y, c='purple', alpha=0.5, s=100)
plt.title('Scatter Plot of Random Points')
plt.xlabel('X values')
plt.ylabel('Y values')
plt.grid(True)
plt.show()
```

Creating Scatter Plots Output



Scatter Plot of Random Points
Y values
X values
2
1
0
−1
−2
−2
−1
0
1
2

More Diagrams?

- There are of course many, many more diagrams available in Matplotlib
  - Histograms, Bar Charts, Pie Chart...
- It is also possible to export diagrams to file (as images)
  - Resolution and DPI can be set as well
- Matplotlib has found its place as the go to solution for diagrams using Python and is often used in scientific contexts such as articles

Project: Yatzy Game 🎲

Yatzy

- The project this year is the game of Yatzy
- It is played using five dice
- The goal is to score the highest number of points matching certain combinations
  - Combinations that are similar, but not exactly, the same as poker
- The dice are rolled up to three times to achieve a scoring combination
  - The player decides what dice to “hold”
- Usually, two or more players take turn in rolling the dice

Rules

- The game is also detailed at https://en.wikipedia.org/wiki/Yatzy
- Also notice that there is a game called *Yahtzee* which is similar but different
  - In this course we want you to implement Yatzy and not Yahtzee...
- There are still variations for the rules within Yatzy
  - If in doubt, decide how it should work and stick to that
- Overall, the small difference in rules that exist are not affecting the complexity of the implementation

Gameplay

1. **Rolling Dice**
    - ➤ Each player rolls five dice
    - ➤ On each turn, the player can roll the dice up to three times
        - ➤ After the first roll, they may choose to keep or “hold” any of the dice and re-roll the others
    - ➤ This can be repeated up to two more times
        - ➤ The player choosing to hold or roll different dice each time

Gameplay, cont.

2. **Choosing a Category**

- After the third roll, the player must select a scoring category
  - Or earlier if the player is satisfied
- Once a category has been chosen and filled, it cannot be used again
- If there is no matching scoring category, or if the player does not want to use a suitable category, it can be crossed out
  - Which means that no points are given for that category

Categories and Scoring

- The score sheet consists of two sections: **Upper Section** and **Lower Section**
- Upper Section
  - In this section, the goal is to get as many of each die face (1-6) as possible
  - The score for each category is the sum of the dice showing that number
    - Three ones give three points and three fives give fifteen points
- Bonus
  - If the total score in the Upper Section is **63 points or more**, the player earns a **50-point bonus**

Score Card

Lower Section, part 1

- In this section, the scores come from specific combinations of dice
- **One Pair**: Two dice showing the same number
  Score: Sum of the two dice
  - *Example*: Roll: 3, 5, 3, 6, 4. Score is **3 + 3 = 6**
- **Two Pairs**: Two pairs of dice showing the same number **Score**: Sum of all four dice
  - *Example*: Roll: 4, 4, 6, 6, 2. Score is **4 + 4 + 6 + 6 = 20**
- **Three of a Kind**: Three dice showing the same number **Score**: Sum of the three dice
  - *Example*: Roll: 5, 5, 5, 2, 6. Score is **5 + 5 + 5 = 15**

Lower Section, part 2

- **Four of a Kind**: Four dice showing the same number **Score**: Sum of the four dice
    - *Example*: Roll: 2, 2, 2, 2, 6. Score is **2 + 2 + 2 + 2 = 8**
- **Small Straight**: A sequence of five consecutive numbers (1-2-3-4-5) **Score**: 15 points
    - *Example*: Roll: 1, 2, 3, 4, 5. Score is **15**
- **Large Straight**: A sequence of five consecutive numbers (2-3-4-5-6) **Score**: 20 points
    - *Example*: Roll: 2, 3, 4, 5, 6. Score is **20**

Lower Section, part 3

- **Full House**: A combination of Three of a Kind and One Pair **Score**: Sum of all five dice
  - *Example*: Roll: 3, 3, 3, 6, 6. Score is **3 + 3 + 3 + 6 + 6 = 21**
- **Chance**: Any combination of dice **Score**: Sum of all five dice
  - *Example*: Roll: 2, 3, 6, 4, 5. Score is **2 + 3 + 6 + 4 + 5 = 20**
- **Yatzy (Five of a Kind)**: All five dice show the same number **Score**: 50 points
  - *Example*: Roll: 6, 6, 6, 6, 6. Score is **50**

Winning the Game

- Once all players have filled in every category, the scores are totalled
- The player with the highest total score, including any bonuses, wins the game
- The highest possible score is **375**
- Anything over 300 is considered a good score
- “Normal” score is somewhere between 150 and 250

Requirements for All Grades

- For the grade E and above, the following needs to be done:
  - A **one player** game of Yatzy
  - Text based user interface that guides the user
  - The possibility of showing the scorecard for the player
  - When rolling, guiding the user to what can be done
    - That is, ask which of the dice to hold
    - Also show what possible positions the dice can be put at

Requirements for Grade C

- All of the previous ones, but also:
  - Support for two or more players in the game
  - High score list of all played games that are written to a file and loaded when the game starts
    - Should of course show the highest score on top

Requirements for Grade A

- All of the previous ones, but also:
  - Error handling for all input
  - Something that will “amaze” us 😆
- It is also possible to implement the somewhat larger Yatzy game called *Maxi Yatzy* for grade A
  - You will need to find the rules yourselves, but they are available on internet

Requirements for Grades D and B

- For grades D or B you need to implement *something* from the higher grades
  - For example, a grade E with error handling can give a D
- The *code quality* is also a factor for a higher (or lower) grade
  - A solution for grade C but with very nice coding could give a B
  - Likewise, a solution for grade C with very poor coding quality could give a D
- You should, when handing in the project, state what grade you are striving for

The Work

- You work in pairs that you decide yourselves, see spreadsheet on Moodle
  - Groups must be decided ASAP
- During next week, the tutoring session will be there to help you with your code and understanding of the task
- The hand-in consists of two parts:
  - A written report handed in November 8, 23:59 (template will be available on Moodle)
  - An oral presentation in week 45

Working as a Team

- For each team we suggest the following:
  - Get started right away. Get in contact with your team members. Establish ways of communication.
  - Daily meetings to give status updates. Progress made, problems faced.
  - A team helps each other. You win (high grade) and lose (fail) as a team.
  - Work together, don’t divide workload in two equal parts.
  - Ask your supervisors if you don’t understand a certain part of the task formulation. Better to ask for the way than to walk in the wrong direction!

Tutoring sessions

- Show progress and discuss next step
- Discuss progress and problems with project supervisor
- Supervisor will not act as project leader dividing work and telling you what to do. Each team leads themselves.
- Remember: Writing the report takes 1-2 days!

End of the Course

- With that, we are at the end of the course
- You have taken your first academic steps to become Python programmers 🐍
- The project will be the final test to see that you understand the foundations