

Packages

Overview

- **Definitions:**
 - What a package is, and how it relates to scripts, modules, and libraries.
- **The Python distribution:**
 - Does not include everything enough
 - when you need extra functionality?
- **Built-in packages:**
 - The import keyword, and simulating randomness with the random module.
- **Installing third-party packages:**
 - pip, PyPI, examples.
- **Conclusions**



Definitions

- What is a 'package'
- Relation to other terms: 'scripts', 'modules', 'libraries', 'functions'

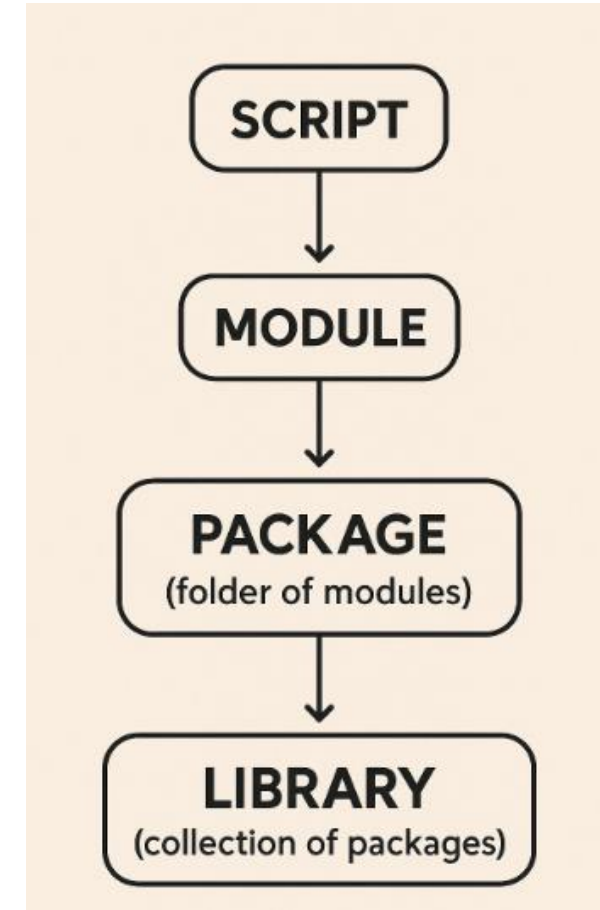
What is a package

- **A collection of code**
 - written by someone else
 - ... or you
 - you can create packages
- **Ready-made 'tools' inside**
 - something to use (implementation detail hidden)
 - ... in your own software creations



Nested relationships

- **Function**
 - small collection of special-purpose code
- **Script**
 - file you run
 - internal structure? (e.g., functions, main)
- **Module**
 - script with several functions
 - imported into other scripts
- **Package**
 - folder-bundled collection of modules
- **Library**
 - collection of packages



The Python distro

- **‘General’ programming lang.**
 - many, many use-cases
- **Light distribution**
 - core functionality
- **Install additional functionality**
 - basic arithmetic is often not enough
 - e.g. no core vector

1. VECTOR & ITS REPRESENTATION Column Vector $\vec{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$ Row Vector $\vec{a} = [a_1 \ a_2 \ a_3]$ Vector in Space $\vec{a} = a_1 \hat{i} + a_2 \hat{j} + a_3 \hat{k}$ $= (a_1, a_2, a_3)$ Unit Vectors $\hat{i} = (1, 0, 0)$, $\hat{j} = (0, 1, 0)$, $\hat{k} = (0, 0, 1)$	2. EQUALITY OF VECTORS Two vectors are equal if they have same magnitude and same direction. $\vec{a} = \vec{b} \Leftrightarrow a_1 = b_1, a_2 = b_2, a_3 = b_3$ Properties $\vec{a} = \vec{b} \Leftrightarrow \vec{a} - \vec{b} = \vec{0}$ $\vec{a} = \vec{0} \Leftrightarrow \vec{a}$ is a zero vector $-\vec{a} = \vec{a}$ with opposite direction $ \vec{a} = -\vec{a} $	3. MAGNITUDE OF A VECTOR For $\vec{a} = a_1 \hat{i} + a_2 \hat{j} + a_3 \hat{k}$ $ \vec{a} = \sqrt{a_1^2 + a_2^2 + a_3^2}$ Unit Vector in the direction of $\vec{a}$ $\hat{a} = \frac{\vec{a}}{ \vec{a} } = \frac{a_1 \hat{i} + a_2 \hat{j} + a_3 \hat{k}}{\sqrt{a_1^2 + a_2^2 + a_3^2}}$ Distance between two points $A(a_1, a_2, a_3)$ and $B(b_1, b_2, b_3)$ $AB = \vec{b} - \vec{a} = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + (b_3 - a_3)^2}$
4. VECTOR ADDITION Triangle Law Parallelogram Law If $\vec{a} = a_1 \hat{i} + a_2 \hat{j} + a_3 \hat{k}$, $\vec{b} = b_1 \hat{i} + b_2 \hat{j} + b_3 \hat{k}$ $\vec{a} + \vec{b} = (a_1 + b_1)\hat{i} + (a_2 + b_2)\hat{j} + (a_3 + b_3)\hat{k}$ $= (a_1 + b_1, a_2 + b_2, a_3 + b_3)$	5. VECTOR SUBTRACTION Triangle Law If $\vec{a} = a_1 \hat{i} + a_2 \hat{j} + a_3 \hat{k}$, $\vec{b} = b_1 \hat{i} + b_2 \hat{j} + b_3 \hat{k}$ $\vec{a} - \vec{b} = (a_1 - b_1)\hat{i} + (a_2 - b_2)\hat{j} + (a_3 - b_3)\hat{k}$ $= (a_1 - b_1, a_2 - b_2, a_3 - b_3)$	6. SCALAR MULTIPLICATION If $\vec{a} = a_1 \hat{i} + a_2 \hat{j} + a_3 \hat{k}$ and λ is a scalar $\lambda \vec{a} = \lambda a_1 \hat{i} + \lambda a_2 \hat{j} + \lambda a_3 \hat{k} = (\lambda a_1, \lambda a_2, \lambda a_3)$ $ \lambda \vec{a} = \lambda \vec{a} $ $\lambda > 0$: $\vec{a}$ and $\lambda \vec{a}$ same direction $\lambda < 0$: $\vec{a}$ and $\lambda \vec{a}$ opposite direction $\lambda = 0$: $\lambda \vec{a} = \vec{0}$
7. DOT PRODUCT (SCALAR PRODUCT) If $\vec{a} = a_1 \hat{i} + a_2 \hat{j} + a_3 \hat{k}$, $\vec{b} = b_1 \hat{i} + b_2 \hat{j} + b_3 \hat{k}$ $\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$ Also, $\vec{a} \cdot \vec{b} = \vec{a} \vec{b} \cos \theta$ Important Points $\vec{a} \cdot \vec{a} = \vec{a} ^2$ $\vec{a} \cdot \vec{b} = 0 \Leftrightarrow \vec{a} \perp \vec{b}$ (Perpendicular) $-1 \leq \cos \theta \leq 1$	8. CROSS PRODUCT (VECTOR PRODUCT) If $\vec{a} = a_1 \hat{i} + a_2 \hat{j} + a_3 \hat{k}$, $\vec{b} = b_1 \hat{i} + b_2 \hat{j} + b_3 \hat{k}$ $\vec{a} \times \vec{b} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$ $= (a_2 b_3 - a_3 b_2)\hat{i} - (a_1 b_3 - a_3 b_1)\hat{j} + (a_1 b_2 - a_2 b_1)\hat{k}$ Properties $\vec{a} \times \vec{b} = -(\vec{b} \times \vec{a})$ $\vec{a} \times \vec{a} = \vec{0}$ If $\vec{a}, \vec{b}$ are parallel then $\vec{a} \times \vec{b} = \vec{0}$ $ \vec{a} \times \vec{b} = \vec{a} \vec{b} \sin \theta$	9. TRIPLE PRODUCT Scalar Triple Product $[\vec{a} \times \vec{b}] \cdot \vec{c}$ = Volume of the parallelepiped formed by $\vec{a}, \vec{b}, \vec{c}$ Vector Triple Product $\vec{a} \times (\vec{b} \times \vec{c}) = \vec{b}(\vec{a} \cdot \vec{c}) - \vec{c}(\vec{a} \cdot \vec{b})$ $(\vec{a} \times \vec{b}) \times \vec{c} = \vec{b}(\vec{a} \cdot \vec{c}) - \vec{a}(\vec{b} \cdot \vec{c})$ Cyclic Permutation $\vec{a} \times \vec{b} + \vec{b} \times \vec{c} + \vec{c} \times \vec{a} = \vec{0}$ $(\vec{a} \cdot \vec{b})\vec{c} + (\vec{b} \cdot \vec{c})\vec{a} + (\vec{c} \cdot \vec{a})\vec{b} = \vec{0}$
10. POSITION VECTOR Position vector of a point $P(x, y, z)$ is $\vec{OP} = x\hat{i} + y\hat{j} + z\hat{k}$ If $A(x_1, y_1, z_1)$ and $B(x_2, y_2, z_2)$ then $\vec{AB} = (x_2 - x_1)\hat{i} + (y_2 - y_1)\hat{j} + (z_2 - z_1)\hat{k}$	11. PROJECTION Scalar Projection of $\vec{a}$ on $\vec{b}$ $\text{comp}_{\vec{b}} \vec{a} = \frac{\vec{a} \cdot \vec{b}}{ \vec{b} } = \vec{a} \cos \theta$ Vector Projection of $\vec{a}$ on $\vec{b}$ $\text{proj}_{\vec{b}} \vec{a} = \left(\frac{\vec{a} \cdot \vec{b}}{ \vec{b} ^2} \right) \vec{b}$	12. LINE & PLANE USING VECTORS Equation of a Line $\vec{r} = \vec{a} + \lambda \vec{b}$ ($\lambda \in \mathbb{R}$) $\vec{a}$: position vector of a point on the line $\vec{b}$: direction vector Equation of a Plane $\vec{r} \cdot \vec{n} = d$ or $a(x - x_1) + b(y - y_1) + c(z - z_1) = 0$ $\vec{n} = a\hat{i} + b\hat{j} + c\hat{k}$: normal vector d : perpendicular distance from origin

The Python distro

- **‘General’ language**
 - many, many use-cases
- **Light distribution**
 - core functionality
- **Install additional functionality**
 - basic arithmetic is often not enough
 - e.g. no core vector

```
1 def vector_add(v1, v2):
2     if len(v1) != len(v2):
3         print("Error: Vectors must have the same length.")
4         return None
5
6     result = []
7     for i in range(len(v1)):
8         result.append(v1[i] + v2[i])
9
10    return result
11
12
13 # Example usage
14 if __name__ == "__main__":
15     v1 = [1, 2]
16     v2 = [4, 5]
17
18     sum_vector = vector_add(v1, v2)
19     print("Vector 1:", v1)
20     print("Vector 2:", v2)
21     print("Sum:", sum_vector)
22
```

```
1 import numpy as np
2
3 # Example usage
4 if __name__ == "__main__":
5     v1 = np.array([1, 2])
6     v2 = np.array([4, 5])
7
8     result = v1 + v2
9     print("Vector 1:", v1)
10    print("Vector 2:", v2)
11    print("Sum:", result)
12
```

Less code, same output, more speed

Built-in packages

`import` keyword

Built-in package example: `random`

The `import` keyword

- `random`
 - Example package
- `import`
 - Sanctions access
- `import random`
 - to every function in random

```
1  import random
2
3
4  def main():
5      ... coin = random.choice(["heads", "tails"])
6      ... print(coin)
7
8
9  if __name__ == "__main__":
10     ... main()
11
```

Using `random`

- **Accessing functions**

- dot `'.'` accessor
- attribute access operator
- pull out a function for use

- **`random.choice`**

- accept, e.g., list of strings
- return a single element selected randomly
- uniform distribution of probability

```
1  import random
2
3
4  def main():
5      ... coin = random.choice(["heads", "tails"])
6      ... print(coin)
7
8
9  if __name__ == "__main__":
10     ... main()
11
```

Namespaces and `import`

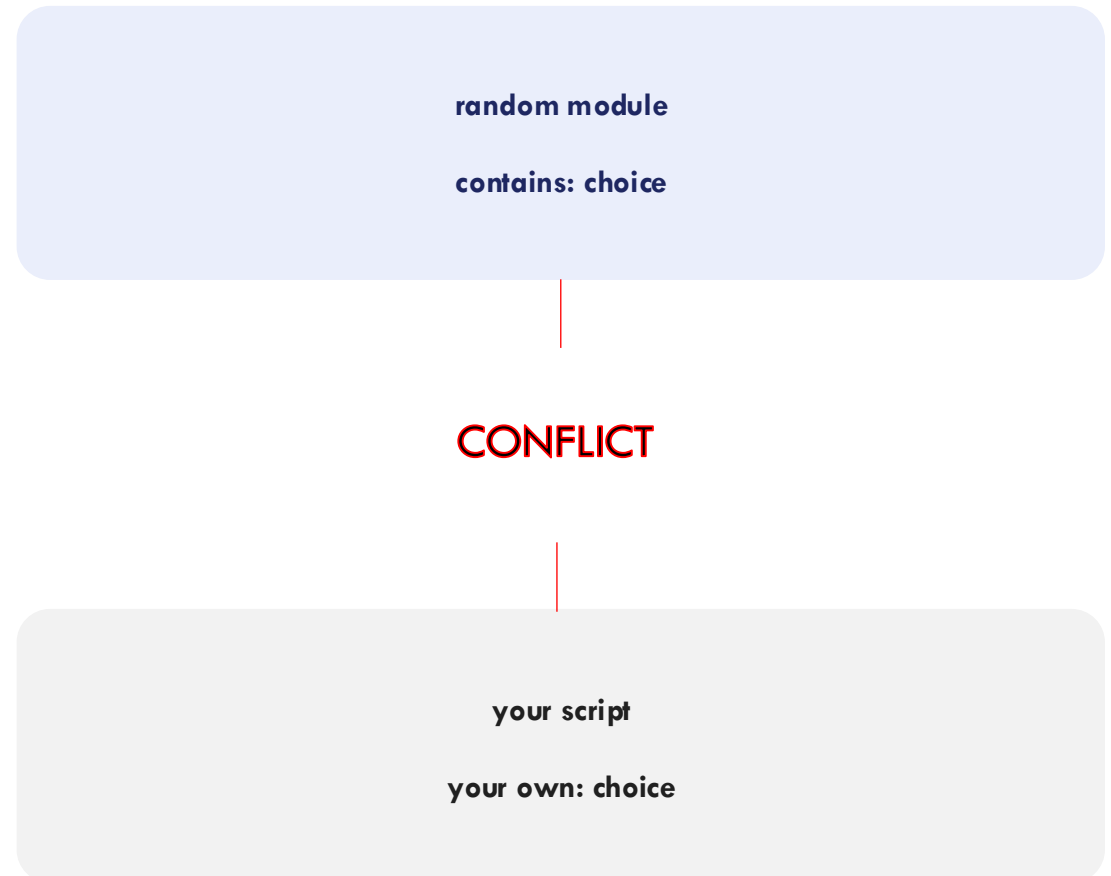
- **Why choose one over the other?**
- **Calling same function often?**
 - `from random import choice`
 - module prefix not needed
- **Using numerous libraries?**
 - module prefix makes code explicit
 - more readable

```
1  import random
2
3
4  def main():
5      coin = random.choice(["heads", "tails"])
6      print(coin)
7
8
9  if __name__ == "__main__":
10     main()
11
```

```
1  from random import choice
2
3
4  def main():
5      print(choice(["heads", "tails"]))
6
7
8  if __name__ == "__main__":
9      main()
10
```

Scope & collision prevention

- **Keeping loaded functions scoped**
 - inside their module container.
- **Namespace Collisions**
 - A local `choice` will overwrite a selectively imported `choice`
- **Scoping Advantages**
 - `random.choice` allows...
 - ...reuse of `choice` identifier locally
 - no 'naming conflicts'
- **Essential...**
 - for large codebases
 - when importing multiple libraries.



Coin toss simulation

- **Idea:**

- `random.randint(0, 1)`
- one of two equally likely outcomes
- State space like a coin

- **Meaning:**

- let 0 mean heads and
- let 1 mean tails.

0 → heads

1 → tails

```
1 import random
2
3
4 def main():
5     coin = random.randint(0, 1)
6
7     if coin == 0:
8         print("heads")
9     else:
10        print("tails")
11
12
13 if __name__ == "__main__":
14     main()
15
```

```
$ python coin.py
heads
$ python coin.py
tails
```

Visualising a coin toss with `turtle`

- **The idea:**

- draw the coin-toss random walk instead of just printing it, using Python's built-in `turtle` module.

- **Start the turtle at**

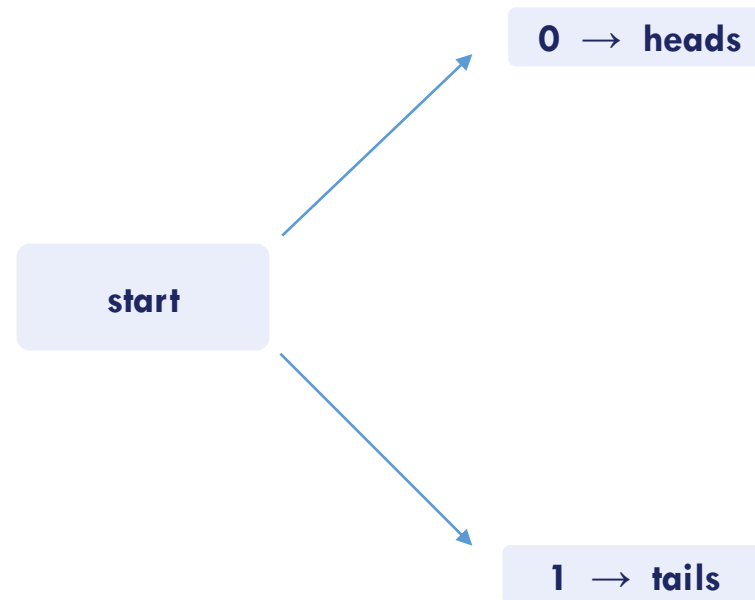
- the left edge of the window
- facing right.

- **Flip a coin at every step**

- heads turns 45° up,
- tails turns 45° down

- **Then move forward**

```
heading = 45 if coin == 0 else -45
```



The result: a wandering path

- **Turtle behaviour**
 - drifts steadily across the window while wobbling up and down
- **Run it again**
 - different path

```
1 import random
2 import turtle
3
4 def main():
5     width = 800
6     height = 600
7     step_size = 20
8     num_steps = 60
9
10    window = turtle.Screen()
11    window.setup(width, height)
12    window.bgcolor("white")
13    window.title("Random Walk")
14
15    walker = turtle.Turtle()
16    walker.shape("turtle")
17    walker.color("#1A73E8")
18    walker.speed(6)
19    walker.penup()
20    walker.goto(-width / 2 + 40, 0)
21    walker.pendown()
22
23    for _ in range(num_steps):
24        coin = random.randint(0, 1)
25        heading = 45 if coin == 0 else 315
26        walker.setheading(heading)
27        walker.forward(step_size)
28
29    window.exitonclick()
30
31 if __name__ == "__main__":
32     main()
33
```



Installing packages

Packages out with core distribution

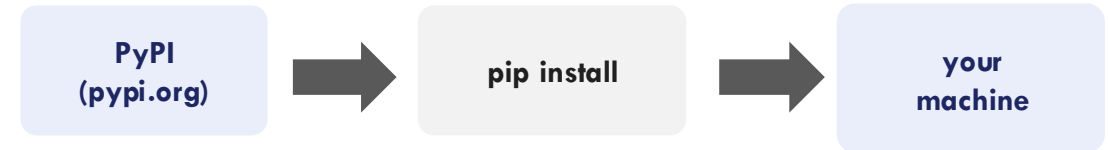
Python **I**nstalls **P**ackages (pip)

Example: `numpy`

Installing packages 'pip'

- **The Python Package Index**

- pypi.org
- massive repository of open-source third-party packages



- **The Package Manager (pip):**

- standard tool to install packages onto your machine

```
$ pip install numpy
```

Example

- **‘General’ language**
 - many, many use-cases
- **Light distribution**
 - core functionality
- **Install additional functionality**
 - basic arithmetic is often not enough
 - e.g. no core vector

```
1 def vector_add(v1, v2):
2     if len(v1) != len(v2):
3         print("Error: Vectors must have the same length.")
4         return None
5
6     result = []
7     for i in range(len(v1)):
8         result.append(v1[i] + v2[i])
9
10    return result
11
12
13 # Example usage
14 if __name__ == "__main__":
15     v1 = [1, 2]
16     v2 = [4, 5]
17
18     sum_vector = vector_add(v1, v2)
19     print("Vector 1:", v1)
20     print("Vector 2:", v2)
21     print("Sum:", sum_vector)
22
```

```
1 import numpy as np
2
3 # Example usage
4 if __name__ == "__main__":
5     v1 = np.array([1, 2])
6     v2 = np.array([4, 5])
7
8     result = v1 + v2
9     print("Vector 1:", v1)
10    print("Vector 2:", v2)
11    print("Sum:", result)
12
```

Less code, same output, more speed

Conclusions

- Packages let you reuse ready-made code instead of writing it yourself.
- Scripts, modules, packages, and libraries each build on the last.
- `import` unlocks the standard library — like `random` — for free.
- `random.randint` powered a coin toss, visualised with a turtle.
- `pip` and PyPI open up the wider Python ecosystem.



Thanks!