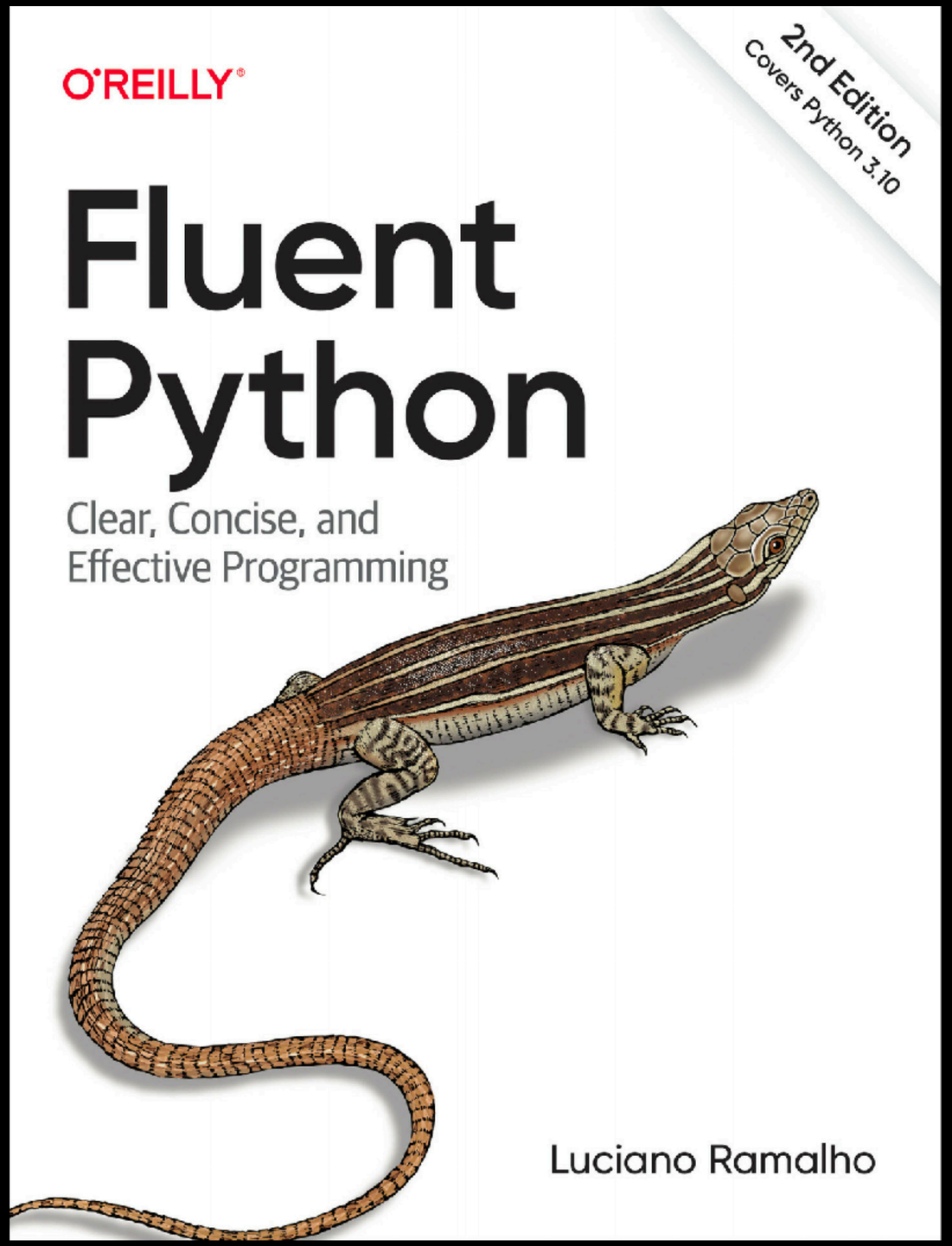
《流畅的Python》

Python进阶教材

演示与导读

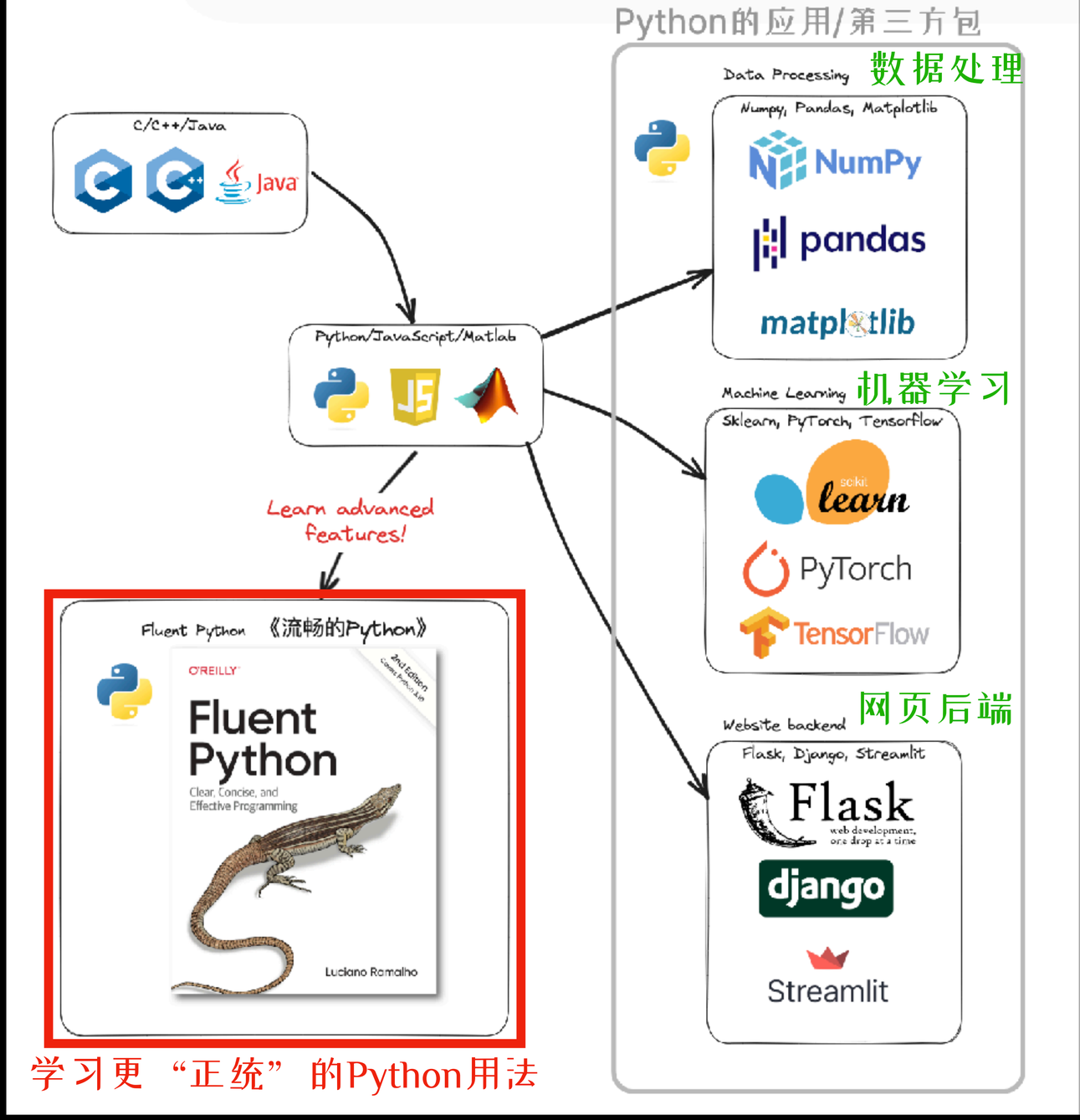


大一上：学习C/C++/Java等语言

大一/大二：

- 使用Python/JavaScript/Matlab动态语言快速解决实际问题
- 根据自己的需求与爱好选择一些项目联手Python，学习第三方包

在此基础上，充分发挥Python语言的动态性/灵活性，你可以进一步提高自己的工作效率/质量.



最基础的语法（以Python为例）

if语句逻辑判断

```
number_a = 33
number_b = 200
if number_b > number_a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

```
b is greater than a
```

最基础的语法（以Python为例）

while循环与for循环

```
i = 1
while i < 6:
    print(i)
    i += 1
```

```
1
2
3
4
5
```

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

```
apple
banana
cherry
```

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes

基本数据类型

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes

基本数据容器/数据结构

使用函数(function)

```
import math
# compute area of circle
def get_circle_area(r):
    return r * r * math.pi

circle_size = get_circle_area(1)
print(circle_size)
```

给定半径，求圆的面积

```
3.141592653589793
```

使用结构体(struct)/类(class)（整合数据与运算逻辑）

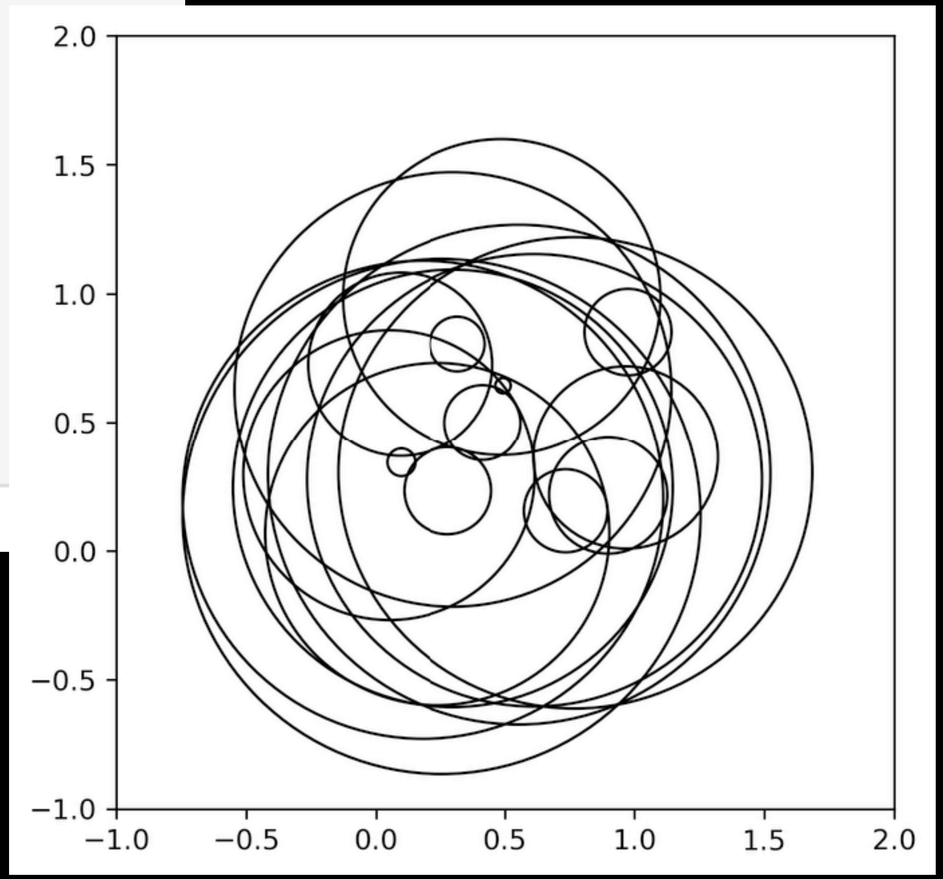
```
class Circle:
    def __init__(self, r, x, y):
        self.r = r
        # the coordinates of circle (x, y)
        self.x = x
        self.y = y

    def get_area(self):
        return math.pi * self.r * self.r

    def get_circumference(self):
        return 2 * math.pi * self.r

    def plot_circle(self):
        return plt.Circle((self.x, self.y), self.r, fill = False)

circles = []
for i in range(20):
    circles.append(Circle(r=random.random(), x=random.random(), y = random.random()))
```

变量:存储数据

函数:存储运算逻辑

例子：建立Circle类，然后随机画20个圆

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 1, figsize=(5, 5), dpi=300)
ax.set_xlim(-1, 2)
ax.set_ylim(-1, 2)

for circle in circles:
    # print(circle.x, circle.y, circle.r)
    ax.add_artist(circle.plot_circle())

plt.show()
```



观点：只要掌握最基本的编程内容，你可以一板一眼地完成很多有意义/有价值的程序

例：Python快速开发（发挥动态语言优势）


```python
class Drone:
    def __init__(self, x, y, v=200):
        self.x = x
        self.y = y
        self.v = v

class Glider:
    def __init__(self, x, y, v=100):
        self.x = x
        self.y = y
        self.v = v

class AirbusPlane:
    def __init__(self, x, y, v=500):
        self.x = x
        self.y = y
        self.v = v
```

打印list列表：Python调用列表中的每个对象(object)所属的类(class)，以及这个对象本身在内存中的16进制地址码

```
plane1 = Drone(0, 20)
plane2 = Glider(0, 100)
plane3 = AirbusPlane(0, 200)

planes = [plane1, plane2, plane3]
```

```
print(planes)
```

```
[<__main__.Drone object at 0x7fc82822c910>, <__main__.Glider object at 0x7fc82822c790>, <__main__.AirbusPlane object at 0x7fc82822cd90>]
```

重载__repr__函数
返回我们想要的
字符串

这里返回的f-string
方便从变量直接构建
字符串


```python
class Drone:
    def __init__(self, x, y, v=200):
        self.x = x
        self.y = y
        self.v = v
    def __repr__(self):
        c_name = self.__class__.__name__
        return f"{c_name} (x,y,v)=({self.x}, {self.y}, {self.v})"

class Glider:
    def __init__(self, x, y, v=100):
        self.x = x
        self.y = y
        self.v = v
    def __repr__(self):
        c_name = self.__class__.__name__
        return f"{c_name} (x,y,v)=({self.x}, {self.y}, {self.v})"

class AirbusPlane:
    def __init__(self, x, y, v=500):
        self.x = x
        self.y = y
        self.v = v
    def __repr__(self):
        c_name = self.__class__.__name__
        return f"{c_name} (x,y,v)=({self.x}, {self.y}, {self.v})"
```

重载__repr__函后print()打印我们希望打印的内容...


```
plane1 = Drone(0, 20)
plane2 = Glider(0, 100)
plane3 = AirbusPlane(0, 200)

planes = [plane1, plane2, plane3]
```

```
for plane in planes:
    print(plane)
```

```
Drone (x,y,v)=(0, 20, 200)
Glider (x,y,v)=(0, 100, 100)
AirbusPlane (x,y,v)=(0, 200, 500)
```

打印一个list列表：Python调用列表中的每个对象的__repr__方法打印内容

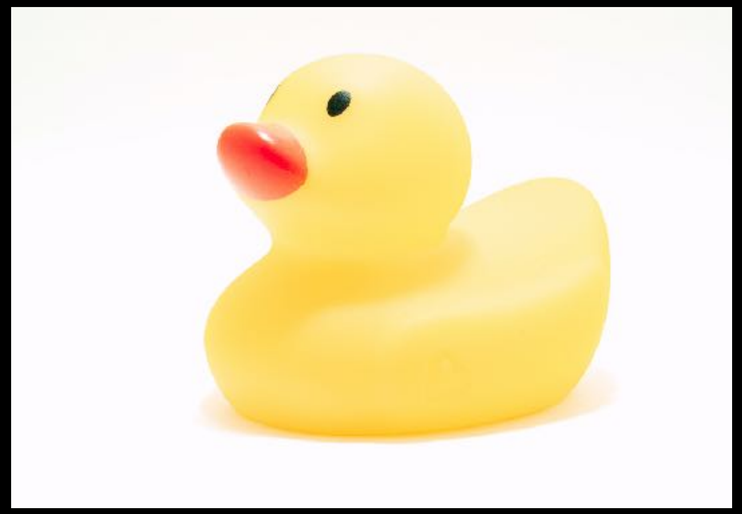
```
# the list know how to print item by item
print(planes)
```

```
[Drone (x,y,v)=(0, 20, 200), Glider (x,y,v)=(0, 100, 100), AirbusPlane (x,y,v)=(0, 200, 500)]
```

Python中部分重载函数(magic methods)

方法名称	被重载的函数/运算符	解释
__repr__ / __str__	print()	打印
__lt__/__gt__/__eq__/__ne__ /...	< / > / == / !=/ ...	比较大小
__id__	hash()	哈希值(set运算)
__add__ / __sub__ / __mul__ /...	+ / - / * /...	数值运算符
__init__ / __del__		创建/删除对象
__abs__ / __neg__ / __round__/__len__	abs() / - / round() / len()	Python中的初始默认函数

所有的Python默认函数(build-in functions)都可以重载



Built-in Functions

A	E	L	R
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
anext()			round()
any()	F	M	
ascii()	filter()	map()	S
	float()	max()	set()
B	format()	memoryview()	setattr()
bin()	frozenset()	min()	slice()
bool()			sorted()
breakpoint()	G	N	staticmethod()
bytearray()	getattr()	next()	str()
bytes()	globals()		sum()
		O	super()
C	H	object()	
callable()	hasattr()	oct()	T
chr()	hash()	open()	tuple()
classmethod()	help()	ord()	type()
compile()	hex()		
complex()		P	V
	I	pow()	vars()
D	id()	print()	
delattr()	input()	property()	Z
dict()	int()		zip()
dir()	isinstance()		
divmod()	issubclass()		_
	iter()		__import__()

duck-typing -- 鸭子类型

指一种编程风格，它并不依靠查找对象类型来确定其是否具有正确的接口，而是直接调用或使用其方法或属性（“看起来像鸭子，叫起来也像鸭子，那么肯定就是鸭子。”）由于强调接口而非特定类型，设计良好的代码可通过允许多态替代来提升灵活性。鸭子类型避免使用 `type()` 或 `isinstance()` 检测。(但要注意鸭子类型可以使用 抽象基类 作为补充。) 而往往会采用 `hasattr()` 检测或是 EAFP 编程。

定义函数order_by_height，返回对象的高度y
列表中的物体根据高度y升序排列

```
# task: sort by their height
def order_by_height(obj):
    return obj.y

print(sorted(planes, key=order_by_height))
```

```
[Drone (x,y,v)=(0, 20, 200), Glider (x,y,v)=(0, 100, 100), AirbusPlane (x,y,v)=(0, 200, 500)]
```

```
sorted(planes, key=lambda obj: obj.y)
```

```
[Drone (x,y,v)=(0, 20, 200),
 Glider (x,y,v)=(0, 100, 100),
 AirbusPlane (x,y,v)=(0, 200, 500)]
```

两种方法，反向排序列表（降序排序）

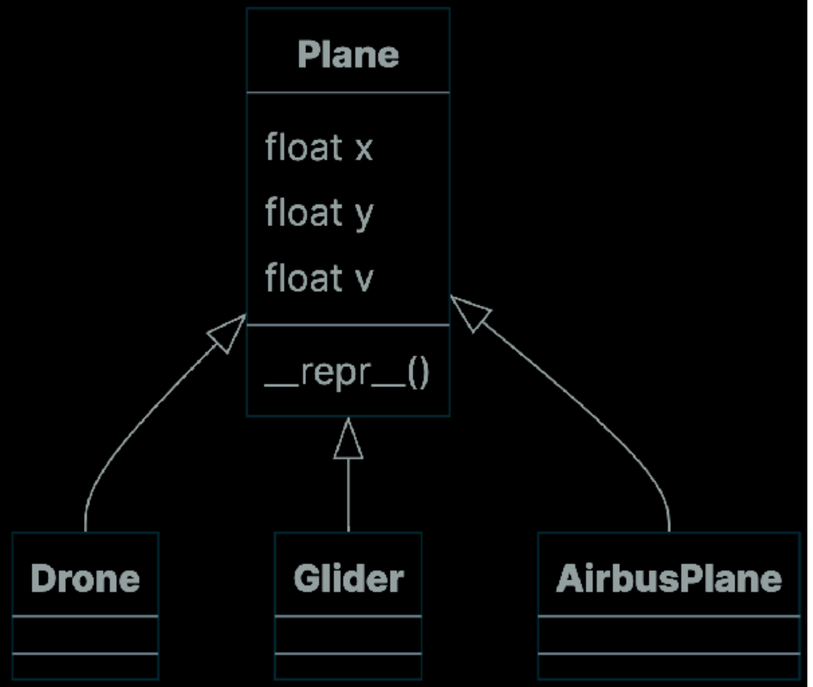
```
sorted(planes, key=lambda obj: obj.y, reverse=True)
```

```
[AirbusPlane (x,y,v)=(0, 200, 500),
 Glider (x,y,v)=(0, 100, 100),
 Drone (x,y,v)=(0, 20, 200)]
```

```
sorted(planes, key=lambda obj: obj.y)[::-1]
```

```
[AirbusPlane (x,y,v)=(0, 200, 500),
 Glider (x,y,v)=(0, 100, 100),
 Drone (x,y,v)=(0, 20, 200)]
```

建立父类（类似的逻辑+避免重复代码）



三个子类继承一个父类

DRY原则：Don't Repeat Yourself

```python
# parent class
class Plane:
    def __init__(self, x, y, v):
        self.x = x
        self.y = y
        self.v = v
    def __repr__(self):
        c_name = self.__class__.__name__
        return f"{c_name} (x,y,v)=({self.x}, {self.y}, {self.v})"


class Drone(Plane):
    def __init__(self, x, y, v=200):
        super().__init__(x, y, x)

class Glider(Plane):
    def __init__(self, x, y, v=100):
        super().__init__(x, y, x)

class AirbusPlane(Plane):
    def __init__(self, x, y, v=500):
        super().__init__(x, y, x)
```

- 目前列表中的对象不能排序
  - 原因：类/对象没有比较定义大小的函数
  - 方案：重载函数
    - def __eq__(self, other): 判断两个物体是否相同
    - def __lt__(self, other): 判断self是否比other小

```
# try to order planes (not successful)
sorted(planes)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/71/d5xhvtkn215cb3g26qsjh5q00000gn/T/ipykernel_12909/1398084992.py in <cell line: 2>()
      1 # try to order planes (not successful)
----> 2 sorted(planes)

TypeError: '<' not supported between instances of 'Glider' and 'Drone'
```

```python
# parent class
class Plane(object):
    def __init__(self, x, y, v):
        self.x = x
        self.y = y
        self.v = v

    def __repr__(self):
        c_name = self.__class__.__name__
        return f"{c_name} (x,y,v)=({self.x}, {self.y}, {self.v})"

    def __eq__(self, other):
        flag = (self.x == other.x) and (self.y == other.y) and (self.v == other.v)
        return flag

    def __lt__(self, other):
        # let's compare by speed
        if self.v < other.v:
            return True
        return False
```

```python
# try to order planes by speed (this time successful)
sorted(planes)
```

更新后运行结果

```
[Glider (x,y,v)=(0, 100, 100),
 Drone (x,y,v)=(0, 20, 200),
 AirbusPlane (x,y,v)=(0, 200, 500)]
```

每个子类定义个性化内容

get_x_max: 估计最远行程

- Drone：使用电池+垂直降落
- Glider：无动力，以高度换行程
- Airbus：烧油+没油后滑翔

```python
class Drone(Plane):
    def __init__(self, x, y, v=200):
        super().__init__(x, y, v)
        self.decline_rate = 0
        self.battery = 100
        self.battery_per_km = 1

    def get_x_max(self):
        travel_dist = (self.battery / self.battery_per_km)
        return self.x + travel_dist


class Glider(Plane):
    def __init__(self, x, y, v=100):
        super().__init__(x, y, v)
        self.decline_rate = 0.05  # decline 1 meter, glide 20 meters

    def get_x_max(self):
        return self.x + self.y / self.decline_rate / 1000


class AirbusPlane(Plane):
    def __init__(self, x, y, v=500):
        super().__init__(x, y, v)
        self.decline_rate = 0.1  # decline 1 meter, glide 10 meters
        self.gas = 100
        self.gas_per_km = 2

    def get_x_max(self):
        travel_dist_gas = self.gas / self.gas_per_km
        travel_dist_glide = self.y / self.decline_rate / 1000
        return self.x + travel_dist_gas + travel_dist_glide
```

不同的类构建的对象使用相同名称的方法/函数都使用get_x_max()估计最大行程

```
plane1 = Drone(0, 200)
plane2 = Glider(0, 5000)
plane3 = AirbusPlane(0, 10_000)

planes = [plane1, plane2, plane3]
print(planes)
```

```
[Drone (x,y,v)=(0, 200, 200), Glider (x,y,v)=(0, 5000, 100), AirbusPlane (x,y,v)=(0, 10000, 500)]
```

```
dists_x_max = [plane.get_x_max() for plane in planes]
print(dists_x_max)
```

```
[100.0, 100.0, 150.0]
```

使用类变量（class variable）

```python
# parent class
class Plane(object):
    num_planes = 0
    def __init__(self, x, y, v):
        self.x = x
        self.y = y
        self.v = v
        Plane.num_planes += 1
```

```python
class AirbusPlane(Plane):
    num_airbus = 0
    def __init__(self, x, y, v=500):
        super().__init__(x, y, v)
        self.decline_rate = 2  # decli
        self.gas = 100
        self.gas_per_km = 10
        AirbusPlane.num_airbus += 1
```

```python
class Drone(Plane):
    num_drones = 0
    def __init__(self, x, y, v=50):
        super().__init__(x, y, v)
        self.decline_rate = 0
        self.battery = 100
        self.battery_per_km = 10
        Drone.num_drones += 1
```

```python
class Glider(Plane):
    num_gliders = 0
    def __init__(self, x, y, v=100):
        super().__init__(x, y, v)
        self.decline_rate = 5  # decl
        Glider.num_gliders += 1
```

该函数随机生成n个飞机对象

```python
import random

def generates_n_planes(n):
    # 0 for drone, 1 for glider, 2 for airbus
    indices = [random.randint(0, 2) for _ in range(n)]

    # create objects
    planes = []
    for ind in indices:
        if ind == 0:
            p = Drone(0, random.gauss(mu=100, sigma=5.0))
        elif ind == 1:
            p = Glider(0, random.gauss(mu=1000, sigma=5.0))
        else:
            p = AirbusPlane(0, random.gauss(mu=5000, sigma=5.0))
        planes.append(p)

    return planes
```

类变量统计100架飞机的中，各种类型飞机的数量

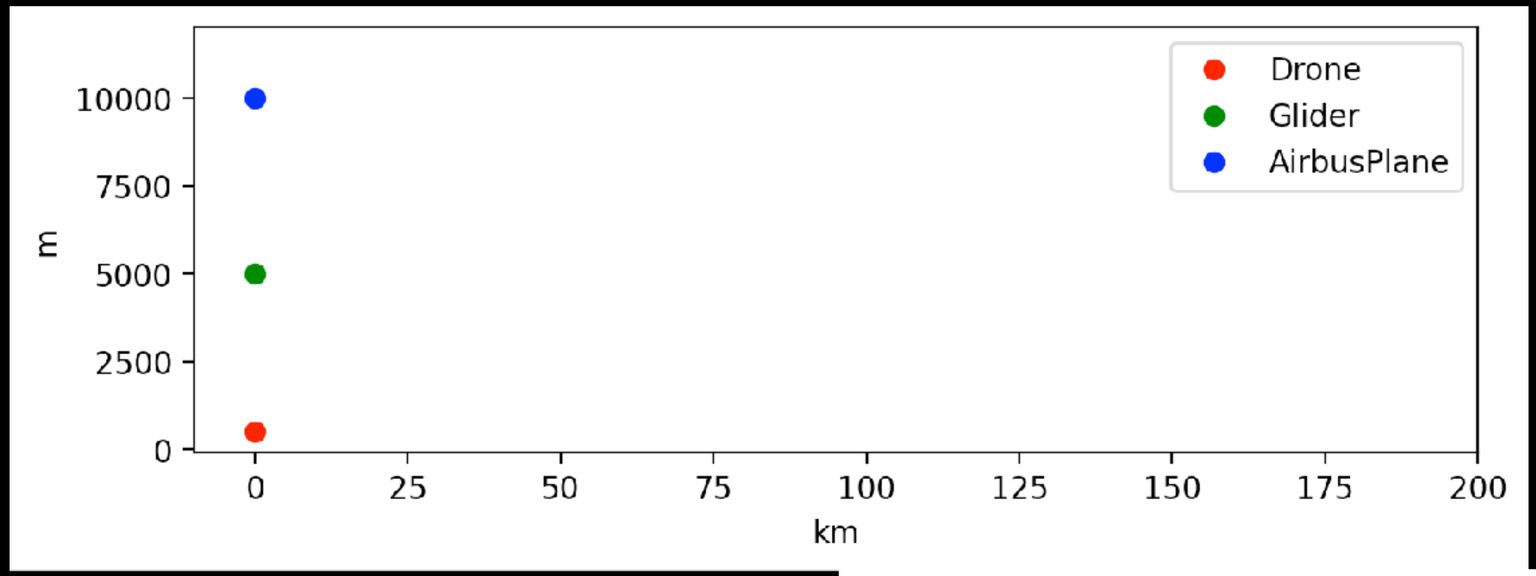
```
# randomly generates 100 planes
planes100 = generates_n_planes(n=100)
print(planes100)
```

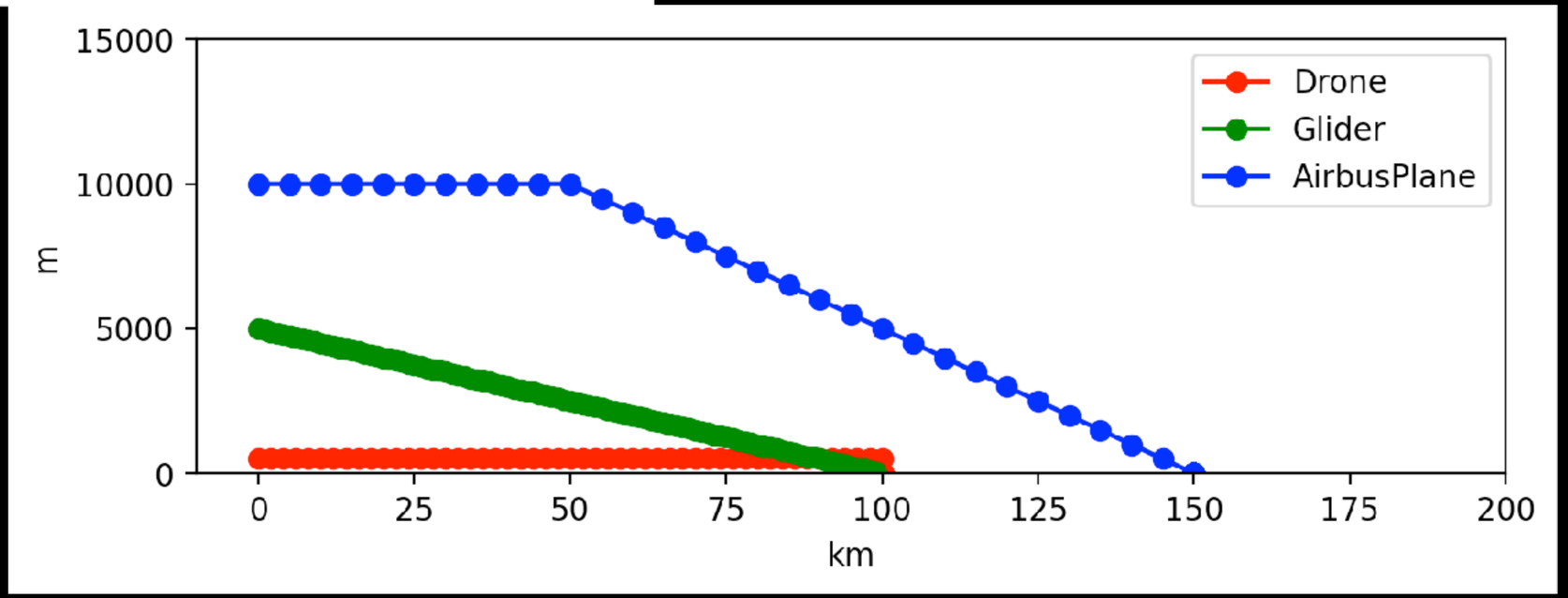
```
[Glider (x,y,v)=(0, 1001.3418515568934, 100), Drone (x,y,v)=(0, 101.28243138117566, 50), Glider (x,y,v)=(0, 1002.9672438105524, 10
0), Glider (x,y,v)=(0, 992.0246505436627, 100), AirbusPlane (x,y,v)=(0, 5000.882672382253, 500), Glider (x,y,v)=(0, 996.874461453006
7, 100), AirbusPlane (x,y,v)=(0, 5013.241941270124, 500), Drone (x,y,v)=(0, 97.51663229523436, 50), Drone (x,y,v)=(0, 92.55762939086
912, 50), Glider (x,y,v)=(0, 996.3747843103558, 100), Glider (x,y,v)=(0, 1003.1173219357006, 100), Drone (x,y,v)=(0, 104.09476754917
84, 50), Glider (x,y,v)=(0, 992.3725963900046, 100), AirbusPlane (x,y,v)=(0, 4995.030829535944, 500), Drone (x,y,v)=(0, 105.92572865
935014, 50), AirbusPlane (x,y,v)=(0, 4992.205790630536, 500), AirbusPlane (x,y,v)=(0, 5002.102886625085, 500), Glider (x,y,v)=(0, 10
02.1271869562028, 100), Glider (x,y,v)=(0, 997.9362594327476, 100), AirbusPlane (x,y,v)=(0, 5002.745142049796, 500), AirbusPlane (x,
y,v)=(0, 4989.628272322785, 500), Drone (x,y,v)=(0, 102.96937826763144, 50), Drone (x,y,v)=(0, 94.71056749856211, 50), Drone (x,y,v)
=(0, 96.33039275101584, 50), Glider (x,y,v)=(0, 990.4009565682923, 100), Drone (x,y,v)=(0, 103.30379994395408, 50), Glider (x,y,v)=
(0, 996.6728410242356, 100), Drone (x,y,v)=(0, 100.43429491546334, 50), AirbusPlane (x,y,v)=(0, 4993.7086840150005, 500), Glider (x,
y,v)=(0, 1006.7420397524543, 100), AirbusPlane (x,y,v)=(0, 4990.041702307531, 500), Drone (x,y,v)=(0, 91.50260211139361, 50), Glider
(x,y,v)=(0, 1002.1984086279618, 100), AirbusPlane (x,y,v)=(0, 4998.253034011059, 500), Drone (x,y,v)=(0, 91.64170076991995, 50), Gli
der (x,y,v)=(0, 993.2065973614991, 100), AirbusPlane (x,y,v)=(0, 4999.835278852114, 500), Drone (x,y,v)=(0, 99.20148386242289, 50),
AirbusPlane (x,y,v)=(0, 4999.462640071139, 5
100), AirbusPlane (x,y,v)=(0, 5000.363589721
3.083070963777, 500), Glider (x,y,v)=(0, 100
```

```
print(f"Total number of planes: {Plane.num_planes}")
print(f"Total number of drones: {Drone.num_drones}")
print(f"Total number of gliders: {Glider.num_gliders}")
print(f"Total number of airbus: {AirbusPlane.num_airbus}")
```

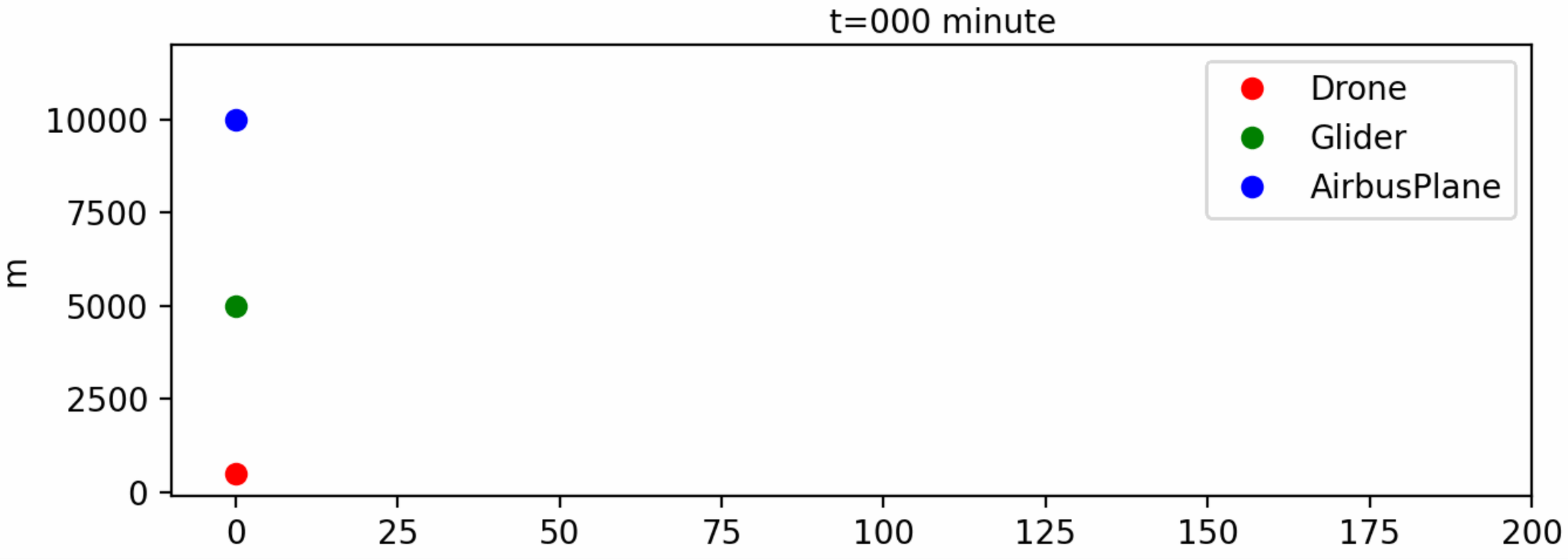
```
Total number of planes: 100
Total number of drones: 27
Total number of gliders: 42
Total number of airbus: 31
```

描写不同飞机的轨迹


Drone
Glider
AirbusPlane
10000
7500
5000
2500
0
m
0
25
50
75
100
125
150
175
200
km

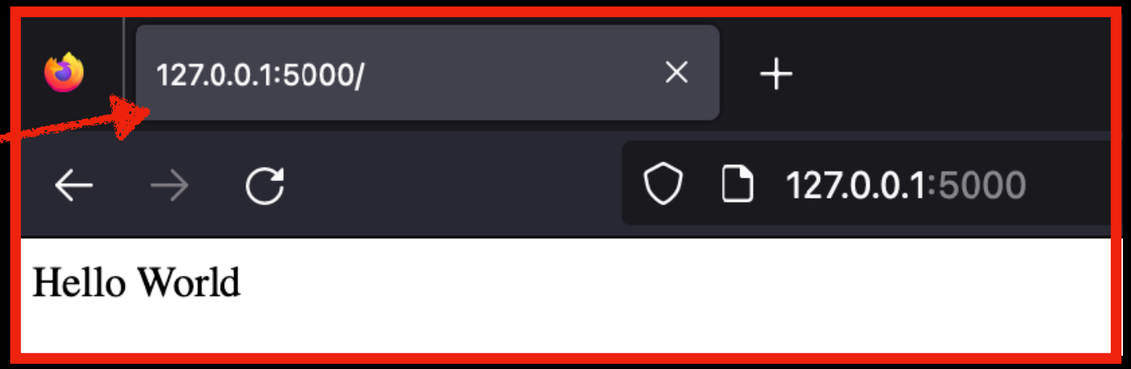

Drone
Glider
AirbusPlane
15000
10000
5000
0
m
0
25
50
75
100
125
150
175
200
km

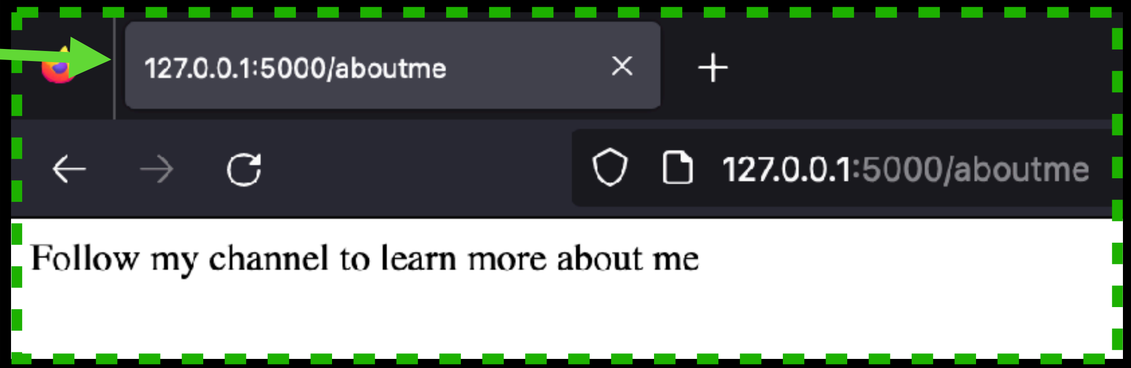
通过gif动画展示不同飞机的飞行方式



修饰器(decorator)
不改变原函数内容基础上添加新的逻辑（如: 注册内容等）
Flask网页后段: 将函数注册到不同网址，返回html网页模版

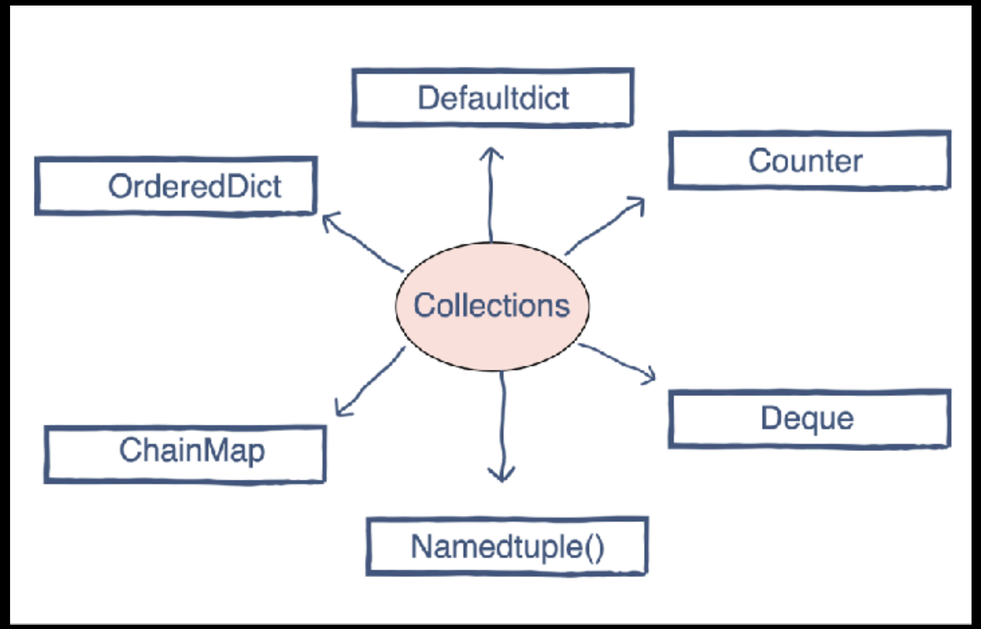
```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World'

@app.route('/aboutme')
def about_me
    return 'Follow my channel to learn more about me'

if __name__ == '__main__':
    app.run()
```





collections模块中更多的数据结构/容器

collection模块中的6种增强数据结构



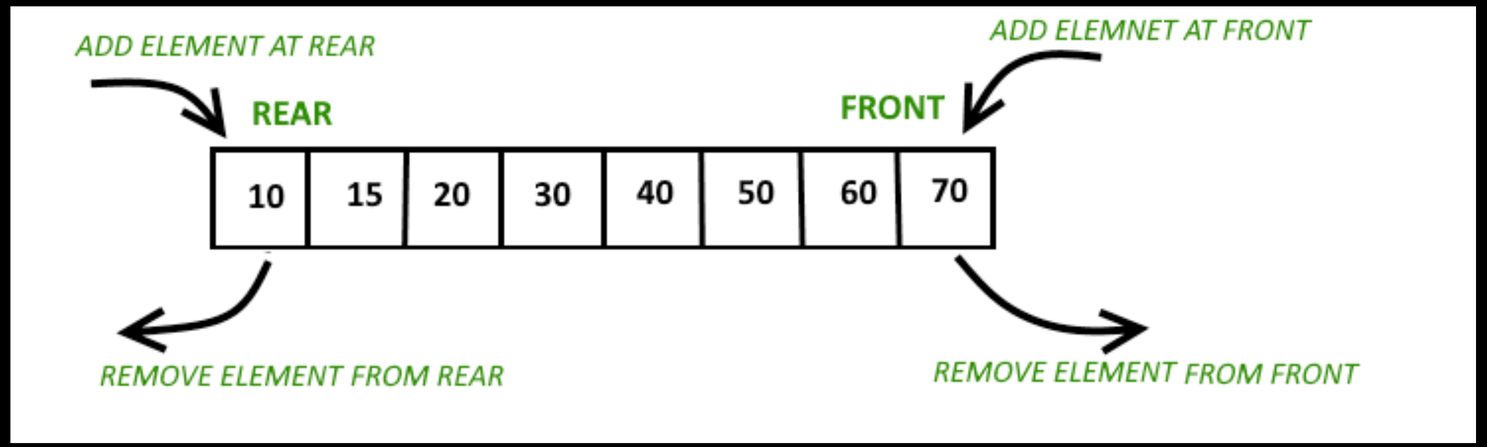
https://www.educative.io/answers/what-is-the-python-collections-module

```python
from collections import OrderedDict
order = OrderedDict()
order['a'] = 1
order['b'] = 2
order['c'] = 3
print(order)

#unordered dictionary
unordered=dict()
unordered['a'] = 1
unordered['b'] = 2
unordered['c'] = 3
print("Default dictionary", unordered)

```

Run

Output

```
OrderedDict([('a', 1), ('b', 2), ('c', 3)])
Default dictionary {'c': 3, 'a': 1, 'b': 2}
```



https://www.geeksforgeeks.org/deque-in-python/

https://www.educative.io/answers/what-is-the-python-collections-module

with语句：环境管理器（Context Manager）

```
# 1) without using with statement
file = open('file_path', 'w')
file.write('hello world !')
file.close()
```

这里在环境
结束时隐藏了
关闭连接的命令

```
# using with statement
with open('file_path', 'w') as file:
    file.write('hello world !')
```

自己实现with环境管理器

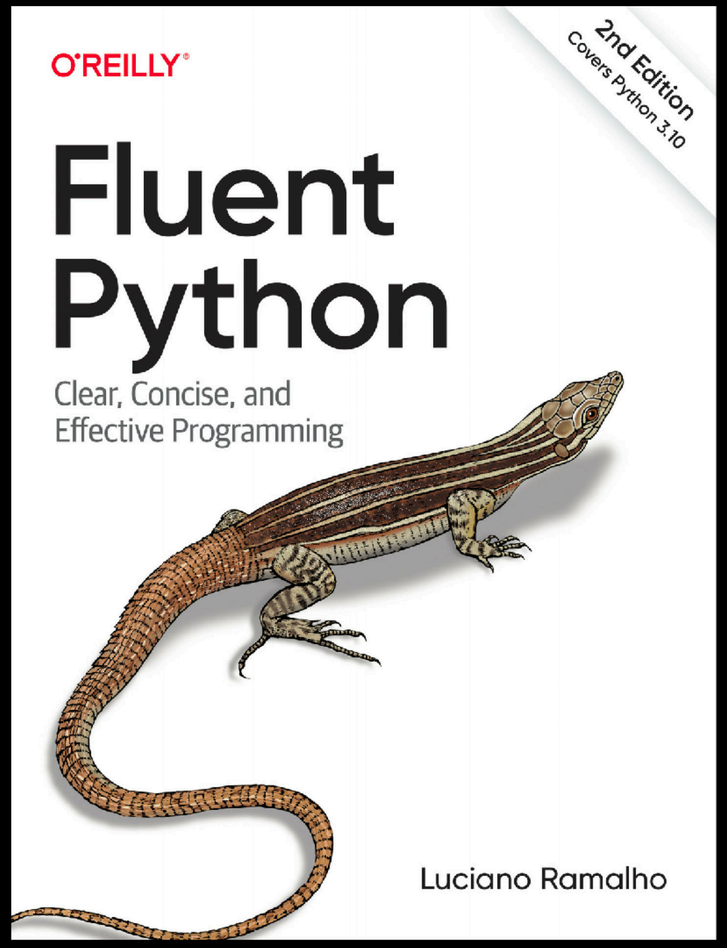
with环境启动执行__enter__中的代码

with环境结束代码__exit__中的代码

```
# a simple file writer object

class MessageWriter(object):
    def __init__(self, file_name):
        self.file_name = file_name

    def __enter__(self):
        self.file = open(self.file_name, 'w')
        return self.file

    def __exit__(self, *args):
        self.file.close()

# using with statement with MessageWriter

with MessageWriter('my_file.txt') as xfile:
    xfile.write('hello world')
```

比较《Fluent Python》两个版本...

第一版封面（2015）
英文版约690页，18章

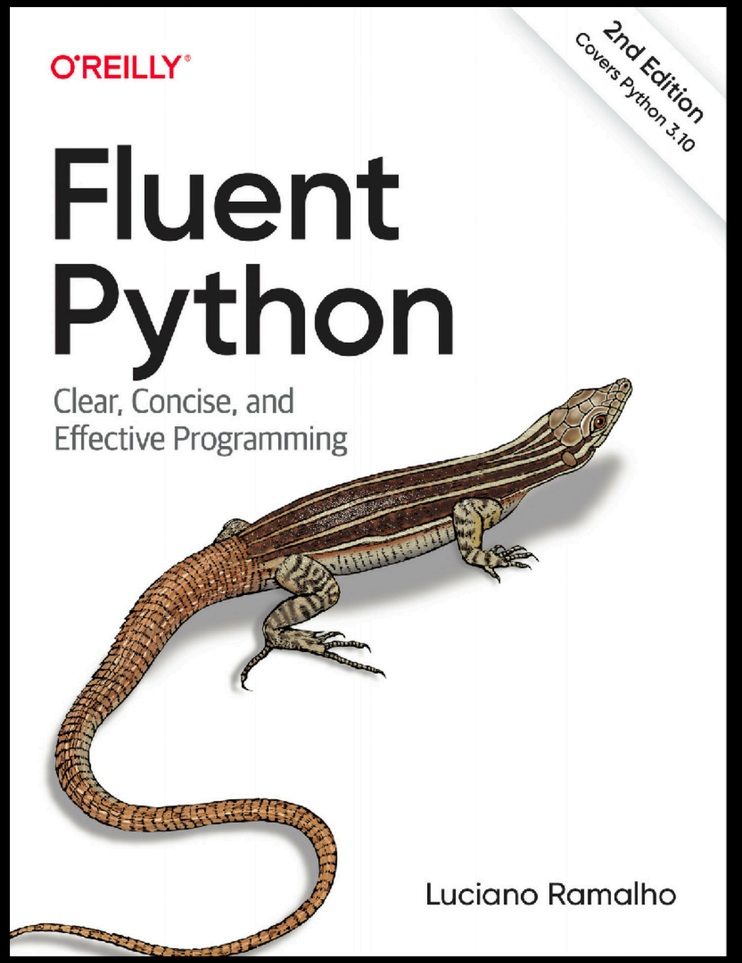


第二版封面（2022）
英文版共962页，24章

Python近期发展更强调规范性内容

- 广泛使用Python支持网络后端的应用的大规模开发

…



第二版封面（2022）
英文版共962页

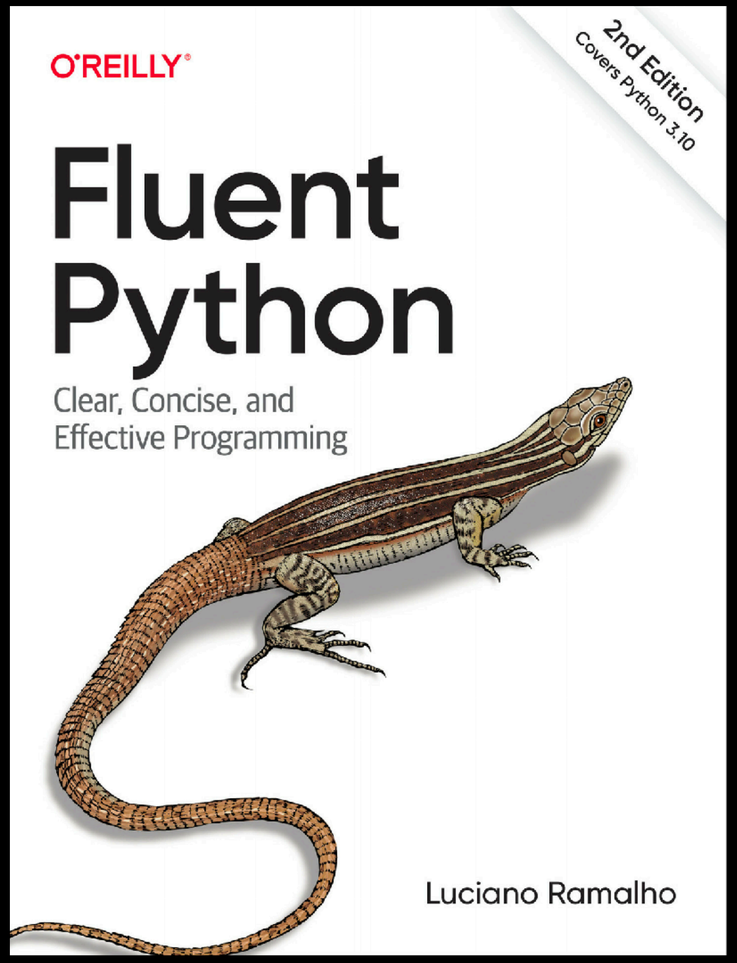
Python近期发展更强调规范性内容

- typing模块变量类型提示（使用Linter提示类型）

```
def show_count(count: int, word: str) -> str:
```

例如函数show_count

- 输入：int类型给变量count
- 输入：str类型给变量word
- 输出：str类型的结果



第二版封面（2022）
英文版共962页

Python近期发展更强调 规范性 内容

- typing模块变量类型提示（使用Linter提示类型）
- abc模块规范类的关系/方法/继承等

@abstractmethod虚拟方法，
子类必须实现

@property属性，
使用getter/setter完成更多逻辑

```python
from abc import ABC, abstractmethod

# parent class
class Plane(ABC):
    def __init__(self, x, y, v):
        self.x = x
        self.y = y
        self.v = v

    def __repr__(self):
        c_name = self.__class__.__name__
        return f"{c_name} (x,y,v)=({self.x}, {self.y}, {sel

    def __eq__(self, other):
        flag = (self.x == other.x) and (self.y == other.y)
        return flag

    def __lt__(self, other):
        if self.v < other.v:
            return True
        return False

    # create abstract methods for subclasses to implement
    @abstractmethod
    def has_fuel(self):
        pass

    @abstractmethod
    def fly_by_hours(self, dt):
        pass

    @property
    def is_on_ground(self):
        if self.y <= 0 :
            return True
        return False
```

I. Data Structures 1. 数据结构

- 1. The Python Data Model
- 2. An Array of Sequences
- 3. Dictionaries and Sets
- 4. Unicode Text Versus Bytes
- 5. Data Class Builders
- 6. Object References, Mutability, and Recycling

II. Functions as Objects 2. 函数（函数即对象）

- 7. Functions as First-Class Objects
- 8. Type Hints in Functions
- 9. Decorators and Closures
- 10. Design Patterns with First-Class Functions

III. Classes and Protocols 3.类与协议接口

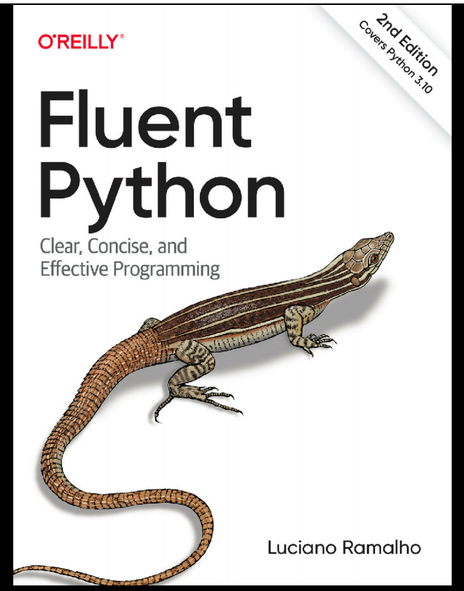
- 11. A Pythonic Object
- 12. Special Methods for Sequences
- 13. Interfaces, Protocols, and ABCs
- 14. Inheritance: For Better or for Worse
- 15. More About Type Hints
- 16. Operator Overloading

IV. Control Flow 4. 程序的流程控制

- 17. Iterators, Generators, and Classic Coroutines
- 18. with, match, and else Blocks
- 19. Concurrency Models in Python
- 20. Concurrent Executors
- 21. Asynchronous Programming

V. Metaprogramming 5.元编程

- 22. Dynamic Attributes and Properties
- 23. Attribute Descriptors
- 24. Class Metaprogramming
- Afterword



免责声明 Disclaimer

- 视频中的部分素材/图片/音乐来源于网络免版权资源。
- 本期视频中的少量代码的例子截图参考自w3schools；geeksforgeeks等网站。
- 视频中的内容仅代表个人的观点，并且尽可能为网友提供一个广泛的，简洁的视角与思路。视频作者无法保证所有观点的绝对严谨与正确。
- 视频中的例子仅作为教学使用，与任何现实中的实际应用或场景没有任何关系。