**1. Конструктор \_\_init\_\_**

def \_\_init\_\_(self, currency, amount):

self.currency = currency

self.amount = amount

* Создаёт объект Currency с двумя атрибутами:
  + currency — тип валюты (например, 'dollar' или 'shekel')
  + amount — количество единиц этой валюты
* Пример:

c1 = Currency('dollar', 5) # объект с 5 долларами

**2. Метод \_\_str\_\_**

def \_\_str\_\_(self):

if self.amount == 1:

return f"{self.amount} {self.currency}"

else:

return f"{self.amount} {self.currency}s"

* Отвечает за **читаемый вывод объекта**, когда используем print() или str(obj).
* Добавляет "s" к названию валюты, если количество больше 1.

Пример:

print(c1) # '5 dollars'

**3. Метод \_\_repr\_\_**

def \_\_repr\_\_(self):

return self.\_\_str\_\_()

* Возвращает **официальное представление объекта**.
* Здесь мы просто делаем его одинаковым с \_\_str\_\_.
* Используется, например, когда выводим объект в списке:

print([c1, c2])

**4. Метод \_\_int\_\_**

def \_\_int\_\_(self):

return self.amount

* Позволяет преобразовать объект в число через int(obj):

print(int(c1)) # 5

**5. Метод \_\_add\_\_**

вызывается при использовании оператора +.

def \_\_add\_\_(self, other):

if isinstance(other, int):

return self.amount + other

elif isinstance(other, Currency):

if self.currency != other.currency:

raise TypeError(f"Cannot add between Currency type <{self.currency}> and <{other.currency}>")

return self.amount + other.amount

else:

return NotImplemented

* Позволяет складывать объект с числом или другой валютой того же типа.
* Если типы валют разные → выбрасывает TypeError.
* Примеры:

print(c1 + 5) # 10

print(c1 + c2) # 15

**6. Метод \_\_iadd\_\_**

When you use x += y, Python will attempt to call x.\_\_iadd\_\_(y).

def \_\_iadd\_\_(self, other):

if isinstance(other, int):

self.amount += other

elif isinstance(other, Currency):

if self.currency != other.currency:

raise TypeError(f"Cannot add between Currency type <{self.currency}> and <{other.currency}>")

self.amount += other.amount

else:

return NotImplemented

return self

* Реализует **операцию +=** (увеличивает текущий объект).
* Примеры:

c1 += 5 # c1.amount = 10

c1 += c2 # c1.amount = 20

**7. Примеры работы**

c1 = Currency('dollar', 5)

c2 = Currency('dollar', 10)

c3 = Currency('shekel', 1)

print(c1) # '5 dollars'

print(int(c1)) # 5

print(repr(c1)) # '5 dollars'

print(c1 + 5) # 10

print(c1 + c2) # 15

c1 += 5

print(c1) # '10 dollars'

c1 += c2

print(c1) # '20 dollars'

* Все операции работают **интуитивно и безопасно**, благодаря проверкам типов и использованию dunder-методов.

№2

## 1. Файл ****func.py****

# func.py

def add\_numbers(a, b):

result = a + b

print(f"The sum of {a} and {b} is {result}")

### Что здесь происходит:

* Определена функция add\_numbers(a, b).
* Она принимает **два аргумента** (a и b).
* Складывает их:
* result = a + b
* Выводит результат в консоль:
* print(f"The sum of {a} and {b} is {result}")

Это форматированная строка (f-string), которая подставляет значения переменных прямо в текст.

## 2. Файл ****exercise\_one.py****

# exercise\_one.py

from func import add\_numbers

add\_numbers(3, 7)

### Что здесь происходит:

* from func import add\_numbers  
  → мы говорим Python: «возьми функцию add\_numbers из файла func.py».  
  Здесь func — это имя файла (без .py).
* add\_numbers(3, 7)  
  → вызываем функцию с аргументами 3 и 7.

## 3. Результат запуска

Когда мы запустим **exercise\_one.py**, в консоли будет:

The sum of 3 and 7 is 10

## 4. Зачем делить на два файла?

* **func.py** — библиотека (здесь лежит функция, которую можно переиспользовать).
* **exercise\_one.py** — программа, которая **импортирует** эту функцию и использует её.

Таким образом, мы тренируемся работать с модулями и импортом в Python.

№3

import string

import random

# Step 2: создать строку из всех букв (верхний и нижний регистр)

letters = string.ascii\_letters # включает 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

# Step 3: сгенерировать случайную строку длиной 5

random\_string = ""

for i in range(5):

random\_char = random.choice(letters) # берём случайную букву

random\_string += random\_char # прибавляем к строке 5 раз

print("Random string:", random\_string)

🔎 Как работает код:

import string, random — подключаем нужные модули.

string.ascii\_letters — строка со всеми маленькими и большими буквами английского алфавита.

nginx

#4

import datetime

def show\_current\_date():

# Step 2: получить текущую дату

today = datetime.date.today()

# Step 3: вывести дату

print("Today's date is:", today)

# Вызов функции

show\_current\_date()

🔎 Объяснение:

import datetime — подключаем модуль datetime.

datetime.date.today() — возвращает текущую дату (год, месяц, день).

print("Today's date is:", today) — выводит её в консоль.

#5

import datetime

def time\_until\_new\_year():

# Step 2: получить текущую дату и время

now = datetime.datetime.now()

# Step 3: создать объект даты для 1 января следующего года

next\_year = now.year + 1

new\_year = datetime.datetime(year=next\_year, month=1, day=1, hour=0, minute=0, second=0)

# Step 4: вычислить разницу

time\_left = new\_year - now

# Step 5: вывести разницу

print("Time left until New Year:", time\_left)

# Вызов функции

time\_until\_new\_year()

🔎 Как работает:

datetime.datetime.now() — текущая дата и время.

datetime.datetime(year=..., month=1, day=1, hour=0, minute=0, second=0) — создаём объект для полуночи 1 января следующего года.

Разница new\_year - now даёт объект timedelta, который показывает дни, часы, минуты и секунды.

#7

import datetime

def minutes\_lived(birthdate\_str):

"""

Function that calculates how many minutes a person has lived.

:param birthdate\_str: birthdate in 'YYYY-MM-DD' format

"""

# Step 1: Parse the input string into a datetime object

birthdate = datetime.datetime.strptime(birthdate\_str, "%Y-%m-%d")

# Step 2: Get the current date and time

now = datetime.datetime.now()

# Step 3: Calculate the time difference

time\_lived = now - birthdate

# Step 4: Convert time lived into minutes

minutes = int(time\_lived.total\_seconds() / 60)

# Step 5: Display the result

print(f"You have lived approximately {minutes} minutes.")

# Example usage

minutes\_lived("1990-05-15")

🔎 Explanation:

datetime.datetime.strptime(birthdate\_str, "%Y-%m-%d")

Converts a string like "1990-05-15" into a datetime object.

datetime.datetime.now()

Gets the current date and time.

now - birthdate

Produces a timedelta object (difference between two dates).

.total\_seconds() / 60

Converts the time difference into minutes.

print(...)

Displays how many minutes the person has lived.

🖥️ Example output:

nginx

Копировать код

You have lived approximately 18420480 minutes.

#7

# Step 2: Import the faker module

from faker import Faker

# Step 3: Create an empty list to store users

users = []

# Step 4: Create a function to add users

def generate\_users(n):

"""

Generate n fake users and store them in the global 'users' list.

Each user is a dictionary with keys: name, address, language\_code.

"""

fake = Faker() # create a Faker instance

for \_ in range(n):

Конструкция for \_ in range(n) в Python означает выполнить блок кода n раз, при этом значение переменной цикла (итерации) не используется внутри цикла. Символ подчёркивания \_ используется как общепринятое соглашение для обозначения того, что значение переменной не нужно и игнорируется, поскольку оно не влияет на логику повторений

user = {

"name": fake.name(),

"address": fake.address(),

"language\_code": fake.language\_code()

}

users.append(user)

# Step 5: Call the function and print the list

generate\_users(5) # generate 5 fake users

for user in users:

print(user)

🔎 Explanation:

from faker import Faker — imports the Faker library.

users = [] — list to store user dictionaries.

fake = Faker() — creates a Faker instance to generate fake data.

The loop runs n times, creating a dictionary for each user:

name → fake.name()

address → fake.address()

language\_code → fake.language\_code()

Append each dictionary to the users list.

Finally, we print all users.