

ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

КАФЕДРА АВТОМАТИЗАЦИЯ ПРОИЗВОДСТВЕННЫХ ПРОЦЕССОВ

ЛЕКЦИЯ №12

Генераторы, элементы
функционального
программирования, аннотации
типов и виртуальная среда в **Python**

СОСТАВИТЕЛЬ: КАНД. ТЕХН. НАУК БЫКАДОР В.С.

Генератор как короткий цикл `for`

`result = [element for element in collection]`

Цикл `for`

```
1
2
3  lst = [1, 30, 2, 4, 2]
4  res = []
5
6  for i in lst:
7      res.append(i**2)
8
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛ

```
• (venv) vityalybykador@Air-Vitaly test % /User
lectures2.py

lst = [1, 30, 2, 4, 2]

res = [1, 900, 4, 16, 4]
```

Генератор

```
3  lst = [1, 30, 2, 4, 2]
4
5  res = [i**2 for i in lst]
6
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛ

```
• (venv) vityalybykador@Air-Vitaly test % /User
lectures2.py

lst = [1, 30, 2, 4, 2]

res = [1, 900, 4, 16, 4]
```

Генератор с условием `if`

`result = [element for element in collection if condition]`

Цикл `for`

```
3  lst = [1, 30, 5, 7, 2]
4  res = []
5
6  for i in lst:
7      if i % 2 == 0:
8          res.append(i)
9
```

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ

```
• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/lectures2.py
```

```
lst = [1, 30, 5, 7, 2]
```

```
res = [30, 2]
```

Генератор

```
3  lst = [1, 30, 5, 7, 2]
4
5  res = [i for i in lst if i % 2 == 0]
6
```

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ ОТЛАДКИ

ТЕ

```
• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/lectures2.py
```

```
lst = [1, 30, 5, 7, 2]
```

```
res = [30, 2]
```

Генератор с расширенным синтаксисом

```
result = [e101, e102 for e101 in collect01 for e102 in collect02]
```

Цикл **for**

```
3  lst_x = [1, 30]
4  lst_y = [4, 2]
5  lst_res = list()
6
7  for x in lst_x:
8      for y in lst_y:
9          lst_res.append(x + y)
10
11
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛА

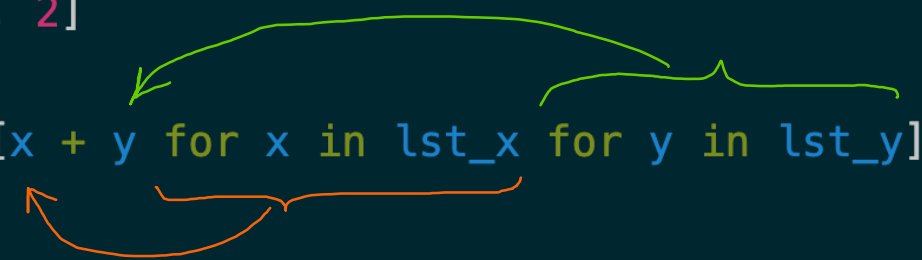
```
• (venv) vitalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/lectures2.py
```

```
lst_x = [1, 30]
lst_y = [4, 2]
```

```
lst_res = [5, 3, 34, 32]
```

Генератор

```
3  lst_x = [1, 30]
4  lst_y = [4, 2]
5
6  lst_res = [x + y for x in lst_x for y in lst_y]
7
8
```



ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ JUPYTER

```
• (venv) vitalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/lectures2.py
```

```
lst_x = [1, 30]
lst_y = [4, 2]
```

```
lst_res = [5, 3, 34, 32]
```

Генератор с расширенным синтаксисом

```
result = [e101, e102 for e101 in collect01 for e102 in collect02 if condition]
```

Цикл **for**

```
3  lst_x = [1, 30]
4  lst_y = [4, 2]
5  lst_res = list()
6
7  for x in lst_x:
8      for y in lst_y:
9          res = x + y
10         if res % 2 == 0:
11             lst_res.append(x + y)
12
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ

```
• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /Use
lectures2.py
```

```
lst_x = [1, 30]
lst_y = [4, 2]
```

```
lst_res = [34, 32]
```

Генератор

```
3  lst_x = [1, 30]
4  lst_y = [4, 2]
5
6  lst_res = [x + y for x in lst_x for y in lst_y if (x + y) % 2 == 0]
7
8
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ JUPYTER

```
• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /Use
lectures2.py
```

```
lst_x = [1, 30]
lst_y = [4, 2]
```

```
lst_res = [34, 32]
```

Сложные генераторы

Примеры с циклом `for`

```
3 lst_x = [1, 3, 5, 4]
4 lst_y = [4, 2, 7, 8]
5 lst_res = []
6
7 for i in range(len(lst_x)):
8     if i % 2 == 0:
9         lst_res.append((lst_x[i] + lst_y[i])**2)
```

частная операция

общая операция

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ JUPYTER

```
• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/lectures2.py
```

```
lst_x = [1, 3, 5, 4]
lst_y = [4, 2, 7, 8]
lst_res = [25, 144]
```

```
3 lst_x = ['q', 'd', 'k', 'f']
4 lst_y = ['i', 'a', 'o', 'e']
5 lst_res = []
6
7 for i in range(len(lst_x)):
8     if i % 2 == 0:
9         lst_res.append((lst_x[i] + lst_y[i])**5)
```

частная операция

общая операция

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ JUPYTER

```
• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/lectures2.py
```

```
lst_x = ['q', 'd', 'k', 'f']
lst_y = ['i', 'a', 'o', 'e']
lst_res = ['qiqiqiqiqi', 'kokokokoko']
```

Вынесем общую операцию в генератор, а частную операцию оставим в цикле.

Сложные генераторы

Предыдущие примеры с генератором

```
2
3 lst_x = [1, 3, 5, 4]
4 lst_y = [4, 2, 7, 8]
5 lst_res = []
6
7
8 def my_gen(lst_x, lst_y):
9     for i in range(len(lst_x)):
10         if i % 2 == 0:
11             yield lst_x[i] + lst_y[i]
12
13     общая операция
14
15 for el in my_gen(lst_x, lst_y):
16     lst_res.append(el**2)
17
18     частная операция
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ

```
• (venv) vitalitybykador@Air-Vitaly test % /Users/vitalybykador/PRO
lectures2.py
```

```
lst_x = [1, 3, 5, 4]
lst_y = [4, 2, 7, 8]
```

```
lst_res = [25, 144]
```

```
○ (venv) vitalitybykador@Air-Vitaly test %
```

```
3 lst_x = ['q', 'd', 'k', 'f']
4 lst_y = ['i', 'a', 'o', 'e']
5 lst_res = []
6
7
8 def my_gen(lst_x, lst_y):
9     for i in range(len(lst_x)):
10         if i % 2 == 0:
11             yield lst_x[i] + lst_y[i]
12
13
14 for el in my_gen(lst_x, lst_y):
15     lst_res.append(el*5)
16
17
```

Генератор

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМ

```
• (venv) vitalitybykador@Air-Vitaly test % /Users/vitalybykad
lectures2.py
```

```
lst_x = ['q', 'd', 'k', 'f']
lst_y = ['i', 'a', 'o', 'e']
```

```
lst_res = ['qiqiqiqiqi', 'kokokokoko']
```

Общие сведения о функциональном программировании

Функциональное программирование представляет собой парадигму программирования, так же как и процедурное программирования и объектно-ориентированное программирование, но в отличии от этих парадигм в функциональном программировании процесс вычисления рассматривается как вычисление значений функций именно в математическом смысле, а не в виде подпрограмм.

Для функционального программирования характерно отсутствие операции присваивания, так как в функциональном программировании нет понятия переменной.

Так как нет переменных для хранения данных, то программа не хранит своё текущее состояние и соответственно, это состояние не может быть изменено из какой-либо операцией, то есть в парадигме функционального программирования нет побочных эффектов.

Функциональное программирование опирается на формальную математическую теорию - **λ -исчисление**.

Что есть в функциональном программировании

Функции высшего порядка. Это такие функции которые могут принимать другие функции в качестве своих аргументов и возвращать функции как результат своей работы.

Чистые функции. Это функции которые не имеют так называемых побочных эффектов, так как данные функции зависят только от своих аргументов.

Что дают чистые функции:

- если чистая функция не используется, то она может быть удалена из программы без вреда для программы;
- результат чистой функции можно мемоизировать, то есть сохранить в таблице в месте с аргументами вызова;
- Если чистые функции не зависят друг от друга, то порядок их вызова можно менять, а так же такие функции можно распараллеливать.

Рекурсия. В функциональном программировании нет понятия цикла, циклы реализуются через рекурсивный вызов функции.

Достоинства и недостатки функционального программирования

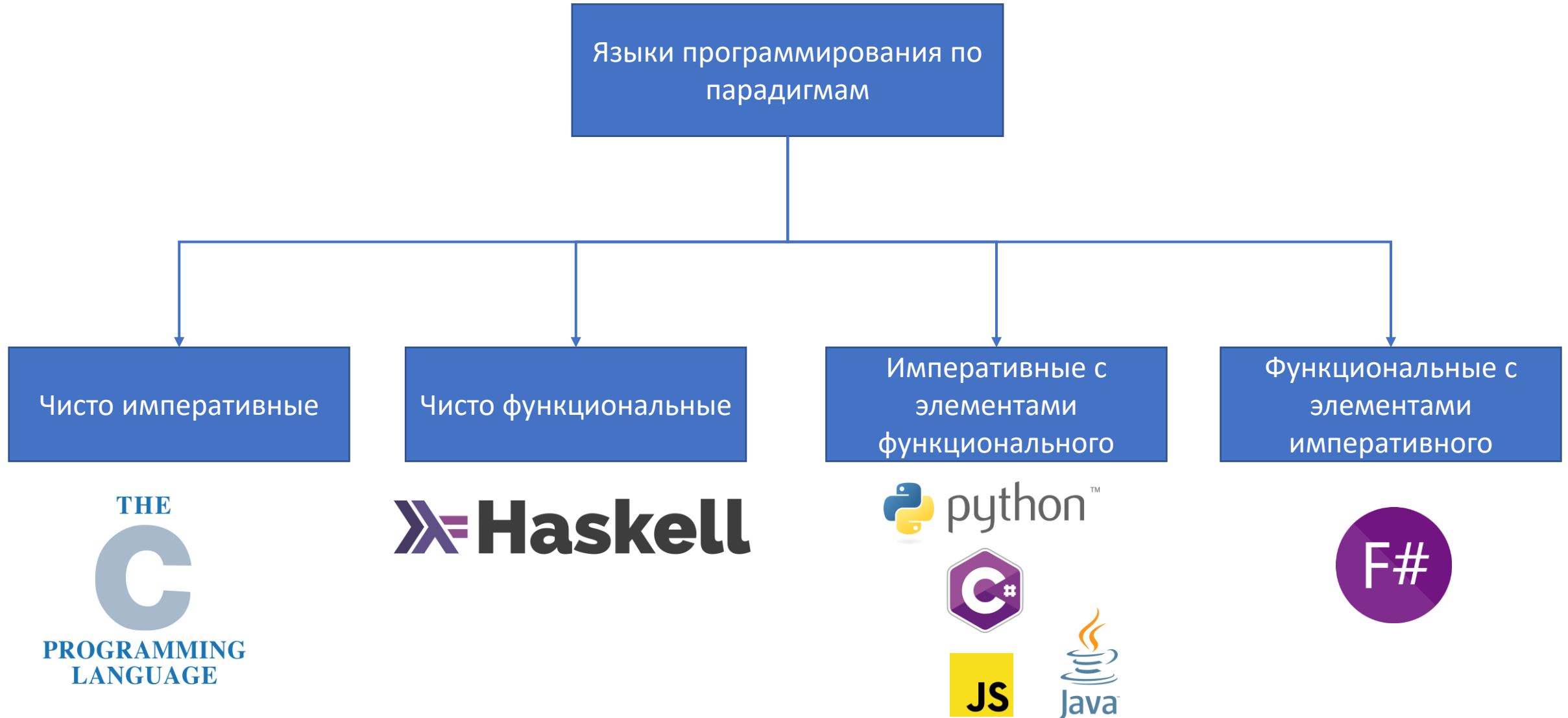
Достоинства:

- Повышение надёжности кода.
- Возможность параллелизма.
- Локальная читаемость кода.
- Возможность оптимизации при компилировании программ.
- Удобство организации модульного тестирования.

Недостатки:

- Высокая степень абстракции функционального программирования не позволяет однозначно определить какие действия будут выполнены конечной аппаратурой => сложно понять сколько времени и памяти потребуется для выполнения программы написанной в функциональном стиле.
- Отсутствие операции присвоения и переменных, а так же наличие рекурсии требует высокоэффективной системы автоматического управления памятью, так называемого «сборщика мусора».
- В рамках функциональной парадигмы сложно реализовать операции ввода-вывода.
- Для тех кто программирует в рамках императивных парадигм может быть сложно понять код программы написанной в функциональной парадигме.

Какие бывают языки программирования



Элементы функционального программирования в Python

Лямбда-выражения (анонимные выражения) – это простые однострочные лямбда-функции (анонимные функции), то есть функции, которые не имеют имени.

`lambda` аргументы: выражение

```
1
2 p = lambda: print('Привет! Я lambda-выражение')
3
4 p()
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ЛУПЫ

```
• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/t
lectures2.py
Привет! Я lambda-выражение ←
○ (venv) vityalybykador@Air-Vitaly test %
```

```
2 p = lambda name: print(f'Привет! Я {name}!')
3
4 p('Толик')
5 p('Ира')
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ЛУПЫ

```
• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJE
lectures2.py
Привет! Я Толик!
Привет! Я Ира!
○ (venv) vityalybykador@Air-Vitaly test %
```

Элементы функционального программирования в Python

Сравните два варианта реализации одной и той же функции.

```
2  def sum(a, b):
3      |   return a + b
4
5  print(f'a + b = {sum(2, 3)}')
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ

- (venv) vityalybykador@Air-Vitaly test % /Users/vit
lectures2.py
a + b = 5
- (venv) vityalybykador@Air-Vitaly test % █

```
2  sum = lambda a, b: a + b
3
4  print(f'a + b = {sum(2, 3)}')
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ

- (venv) vityalybykador@Air-Vitaly test % /Users/vit
lectures2.py
a + b = 5
- (venv) vityalybykador@Air-Vitaly test % █

Элементы функционального программирования в Python

Пример простого калькулятора на основе словаря (из лекции № 03)

```
5 def summ(a, b):
6     return a + b
7
8 def sub(a, b):
9     return a - b
10
11 def mul(a, b):
12     return a * b
13
14 def div(a, b):
15     if b != 0:
16         return a / b
17     else:
18         return 'Ошибка! Деление на ноль.'
19
```

```
19
20 operation = {
21     '+': summ,
22     '-': sub,
23     '*': mul,
24     '/': div
25 }
26
27
28 a, b = 2, 3
29 op = '+'
30
31 if op in operation.keys():
32     func = operation[op]
33     result = func(a, b)
34 else:
35     result = 'Математическая операция не реализована.'
36
37 print(f'result = {result}')
38
```



```
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  JUPYTER

• (venv) vitalitybykador@MacBook-Air-Vitaly test % /Users/vitalybykador/PROJE

result = 5
○ (venv) vitalitybykador@MacBook-Air-Vitaly test %
```

Элементы функционального программирования в Python

Лямбда-выражения

```
2 operation = {
3     '+': lambda a, b: a + b,
4     '-': lambda a, b: a - b,
5     '*': lambda a, b: a * b,
6     '/': lambda a, b: a / b if b != 0 else 'Ошибка! Деление на ноль.'
7 }
8
9 a, b = 2, 3
10 op = '+'
11
12 if op in operation.keys():
13     func = operation[op]
14     result = func(a, b)
15 else:
16     result = 'Математическая операция не реализована.'
17
```

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ ОТЛАДКИ

ТЕРМИНАЛ

JUPYTER

```
● (venv) vitalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /Users/vitalybykador/PROJECTS/test/venv/bin/python /Users/vitalybykador/PROJECTS/test/venv/bin/python lectures2.py
```

```
result = 5
```

Элементы функционального программирования в Python

Функция `map`

`map` (лямбда-выражение: последовательность)

```
1
2 lst = [1, 4, 5, 11, 6]
3 res = []
4
5 for el in lst:
6     res.append(el**2)
7
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ

```
• (venv) vityalybykador@Air-Vitaly test %  
lectures2.py
```

```
res = [1, 16, 25, 121, 36]
```

```
1
2 lst = [1, 4, 5, 11, 6]
3 res = list( map(lambda el: el**2, lst) )
4
5
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ

```
• (venv) vityalybykador@Air-Vitaly test % /Users/vityalybykador/P  
lectures2.py
```

```
res = [1, 16, 25, 121, 36]
```

```
1
2 lst = [1, 4, 5, 11, 6]
3 res = [el**2 for el in lst]
4
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ

```
• (venv) vityalybykador@Air-Vitaly test % /Users/vit  
lectures2.py
```

```
res = [1, 16, 25, 121, 36]
```


Элементы функционального программирования в Python

Функция `filter`

`filter` (лямбда-выражение: последовательность)

```
2  lst = [1, 4, 5, 11, 6]
3  res = []
4
5  for el in lst:
6      if el % 2 == 0:
7          res.append(el)
8
9
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ

• (venv) vitalitybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/lectures2.py

res = [4, 6]

```
1
2  lst = [1, 4, 5, 11, 6]
3  res = list( filter(lambda el: el % 2 == 0, lst) )
4
```

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ ОТЛАДКИ

ТЕРМИНАЛ

JUPYTER

• (venv) vitalitybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/lectures2.py

res = [4, 6]

Элементы функционального программирования в Python

```
1
2  lst = [1, 4, 5, 11, 6]
3  res = []
4
5  for el in lst:
6      if el % 2 == 0:
7          res.append(el**2)
8
```

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ О


● (venv) vitalybykador@Air-Vitaly test % /Use
lectures2.py

res = [16, 36]

```
2  lst = [1, 4, 5, 11, 6]
3  res = list( map(lambda el: el**2 if el % 2 == 0 else ?, lst) )
4
```

Элементы функционального программирования в Python

Замена последовательности действий суперпозицией функций.




```
1
2 lst = [1, 4, 5, 11, 6]
3 res = []
4
5 for el in lst:
6     if el % 2 == 0:
7         res.append(el**2)
8
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ О

• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /Users/vita lectures2.py

res = [16, 36]



```
1
2 lst = [1, 4, 5, 11, 6]
3 res = list( map(lambda el: el**2, filter(lambda el: el % 2 == 0, lst)) )
4
5
```

Возвращает список отфильтрованных элементов

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ JUPYTER

• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /Users/vita lectures2.py

res = [16, 36]

Элементы функционального программирования в Python

Функция `reduce`

reduce (лямбда-выражение: последовательность)

```
1
2 lst = [1, 4, 5, 7, 9]
3 res = sum(lst)
4
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ

```
• (venv) vityalybykador@Air-Vitaly test % python3 lectures2.py
```

```
res = 26
```

```
1 import functools as ft
2
3 lst = [1, 4, 5, 7, 9]
4 res = ft.reduce(lambda x, y: x + y, lst)
5
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ

```
• (venv) vityalybykador@Air-Vitaly test % python3 /Users/vityalybykador/PycharmProjects/lectures2.py
```

```
res = 26
```

$$1 + 4 + 5 + 7 + 9 = 26$$

Элементы функционального программирования в Python

Функция `reduce`

```
1
2 lst = [1, 4, 5, 7, 9]
3 summa = lst[0]
4
5 for i in range(len(lst)):
6     if i > 0:
7         summa += 2*lst[i]
8
```

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ ОТ

```
• (venv) vityalybykador@Air-Vitaly test % /Use
lectures2.py
```

```
res = 51
```

```
1 import functools as ft
2
3 lst = [1, 4, 5, 7, 9]
4 summa = ft.reduce(lambda x, y: x + 2*y, lst)
5
```

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ ОТЛАДКИ

ТЕРМИНАЛ

```
• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJ
lectures2.py
```

```
res = 51
```

$$1 + 2*4 + 2*5 + 2*7 + 2*9 = 51$$

Элементы функционального программирования в Python

Функции `reduce` и `filter`

```
1  
2 lst = [1, 4, 5, 7, 9]  
3 somma = 0  
4  
5 for el in lst:  
6     if el % 2 == 1:  
7         somma += el  
8
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ

• (venv) vitalitybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /Users/vitalybykador/PROJECTS/test/venv/bin/python lectures2.py

res = 22

```
1 import functools as ft  
2  
3 lst = [1, 4, 5, 7, 9]  
4 somma = ft.reduce(lambda x, y: x + y, filter(lambda el: el % 2 == 1, lst))  
5  
6
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ JUPYTER: VARIABLES

• (venv) vitalitybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /Users/vitalybykador/PROJECTS/test/venv/bin/python lectures2.py

res = 22

$$1 + \cancel{4} + 5 + 7 + 9 = 22$$

Аннотации типов

Язык программирования **Python**, как известно, имеет динамическую типизацию и это бывает удобно в плане унификации кода, скорости написания кода и отсутствия необходимости выбора того или иного типа, так же динамическая типизация облегчает реализацию полиморфизма в ООП.

```
1
2 def sum(a, b):
3     return a + b
4
5 a_, b_ = 3, 2
6
7 print()
8 print(f'sum = {sum(a_, b_)}')
9 print()
10
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ

• (venv) vityalybykador@Air-Vitaly test % /Users/vit
or/PROJECTS/test/for_lectures2.py

sum = 5

```
1
2 def sum(a, b):
3     return a + b
4
5 a_, b_ = 'При', 'вет!'
6
7 print()
8 print(f'sum = {sum(a_, b_)}')
9 print()
10
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ

• (venv) vityalybykador@Air-Vitaly test % /Users/vit
or/PROJECTS/test/for_lectures2.py

sum = Привет!

Аннотации типов

Тем не менее в современном **Python** есть возможность указать типы ожидаемых переменных.

Это не означает, что если параметры функции аннотированы типом **int**, и передать этой функции параметры типа **string**, но **PVM** не выполнить код – **PVM** код выполнить.

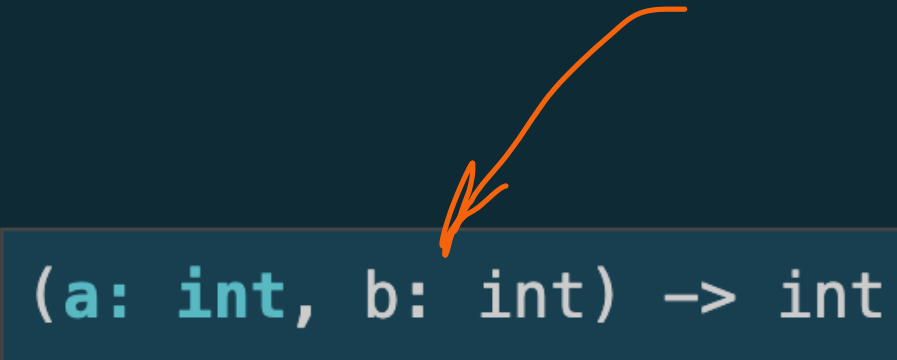
Аннотирование типов необходимо для подсказки разработчику, какого типа параметры функция ожидает.

Так же есть специальные инструменты, например **mypy**, которому можно передать в качестве параметра скрипт на языке **Python** для проверки соответствия передаваемых и получаемых типов параметров в функцию. Это своего рода аналог проверки типов компиляторами в языках со статической типизацией. Но в отличие от компиляторов, **mypy** просто выдаёт предупреждение, а решение исправлять код или нет остаётся за программистом.

Аннотации типов

def name_func(a: name_type, b: name_type) -> name_return_type:

```
1
2  def sum(a: int, b: int) -> int:
3      |    return a + b
4
5  a_, b_ = 2, 3
6
7  print()
8  print(f'sum = {sum()}')
9  print()
10
```



The diagram illustrates the relationship between a function signature and its use. An orange arrow points from the signature `(a: int, b: int) -> int` (highlighted in a box on line 2) to the function call `sum()` (highlighted in a box on line 8). A vertical red line connects the opening parenthesis of `sum()` to the opening parenthesis of the signature, indicating the mapping of arguments.

Аннотации типов

```
1
2 def sum(a: int, b: int) -> int:
3     | return a + b
4
5 a_, b_ = 2, 3
6
7 print()
8 print(f'sum = {sum(a_, b_)}')
9 print()
10
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ

• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/for_lectures2.py

sum = 5

```
1
2 def sum(a: int, b: int) -> int:
3     | return a + b
4
5 a_, b_ = 'aa', 'bb'
6
7 print()
8 print(f'sum = {sum(a_, b_)}')
9 print()
10
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ

• (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/for_lectures2.py

sum = aabb

АННОТАЦИИ ТИПОВ

```
2  class SomeClass:
3      def __init__(self) -> None:
4          self.__lst: list[str] = []
5
6      def add(self, element: str) -> None:
7          self.__lst.append(element)
8
9      @property
10     def data(self) -> list[str]:
11         return self.__lst
12
13
14  obj = So(element: str) -> None
15  obj.add()
```

Аннотации типов

```
2  class SomeClass:
3      def __init__(self) -> None:
4          self.__lst: list[str] = []
5
6      def add(self, element: str) -> None:
7          self.__lst.append(element)
8
9      @property
10     def data(self) -> list[str]:
11         return self.__lst
12
13
14     obj = SomeClass()
15     obj.add('hello')
16     obj.add('всем')
17     obj.add('ребята!')
18     (property) data: list[str]
19     print(obj.data)
```

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ ОТЛАДКИ

ТЕРМИ

```
(venv) vitalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/for_lectures2.py
```

```
['hello', 'всем', 'ребята!']
```

АННОТАЦИИ ТИПОВ

Более подробно про аннотации типов можно узнать в справке по языку **Python**.

<https://docs.python.org/3/library/typing.html#corresponding-to-types-in-collections>

Table of Contents

typing — Support for type hints

- Relevant PEPs
- Type aliases
- NewType
- Callable
- Generics
- User-defined generic types
- The **Any** type
- Nominal vs

typing — Support for type hints

New in version 3.5

Source code: [Lib/typing.py](#)

Note: The Python typing annotations. The typing module, etc.

Type aliases

A type alias is defined by assigning the type to the alias. In this example, `Vector` and `list[float]` will be treated as interchangeable synonyms:

```
Vector = list[float]
```

```
def scale(scalar: float, vector: Vector) -> Vector:  
    return [scalar * num for num in vector]
```

```
# passes type checking; a list of floats qualifies as a Vector.  
new_vector = scale(2.0, [1.0, 2.0, 3.0])
```

```
from typing import NewType
```

```
UserId = NewType('UserId', int)  
some_id = UserId(524313)
```

Виртуальная среда выполнения

Виртуальная среда выполнения позволяет в проекте на **Python** иметь копию необходимых библиотек и **PVM**. Таким образом можно получить полностью изолированный проект и не зависит от обновления библиотек и версий **Python** .


В общем случае процедура создания виртуальной среды заключается в выполнении следующих действий:

Виртуальная среда выполнения


New Project

Location:

▼ Python Interpreter: New Virtualenv environment

☒ New environment using  Virtualenv ▼

Location:

Base interpreter:  ▼ ...

☐ Inherit global site-packages

☐ Make available to all projects

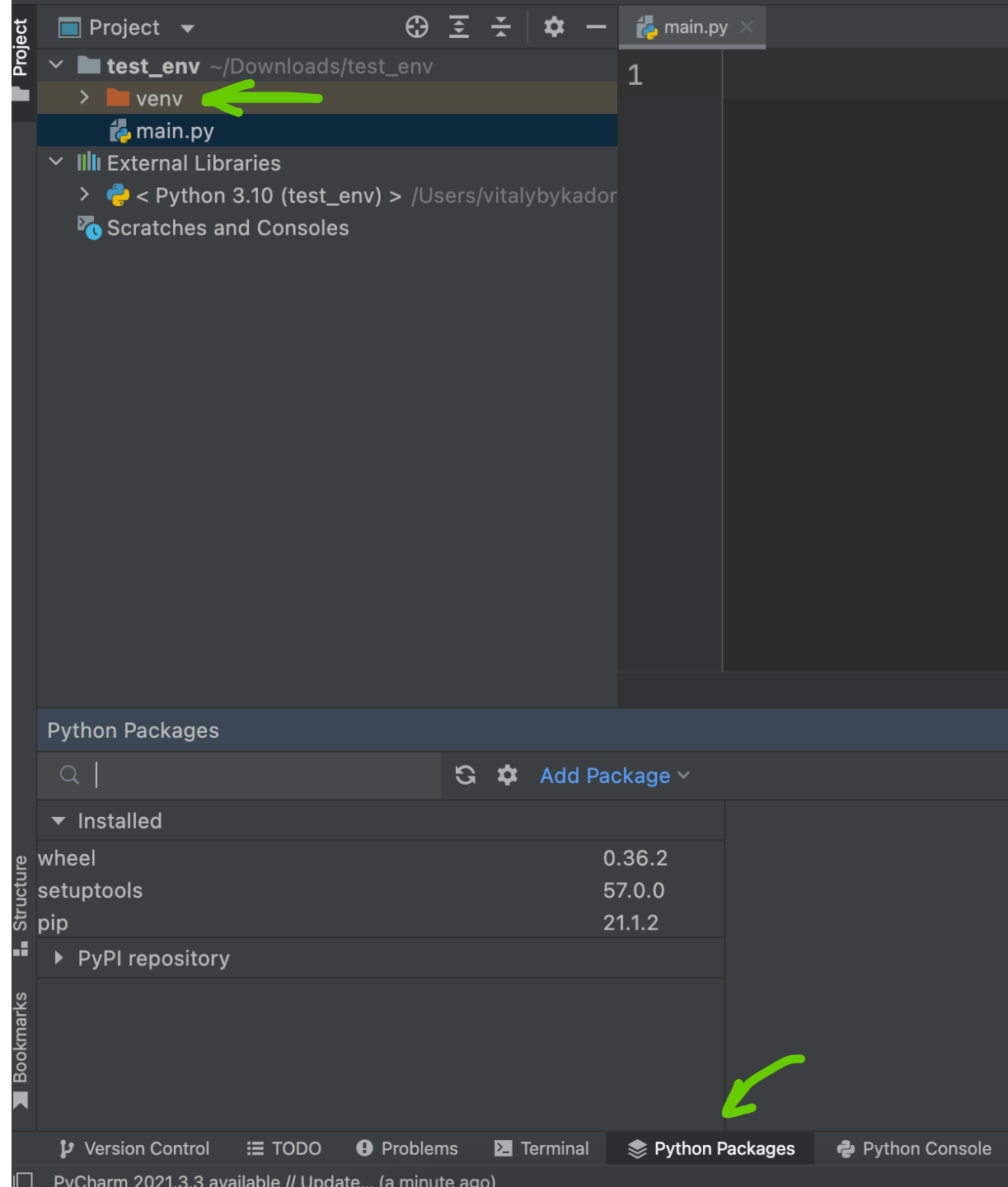
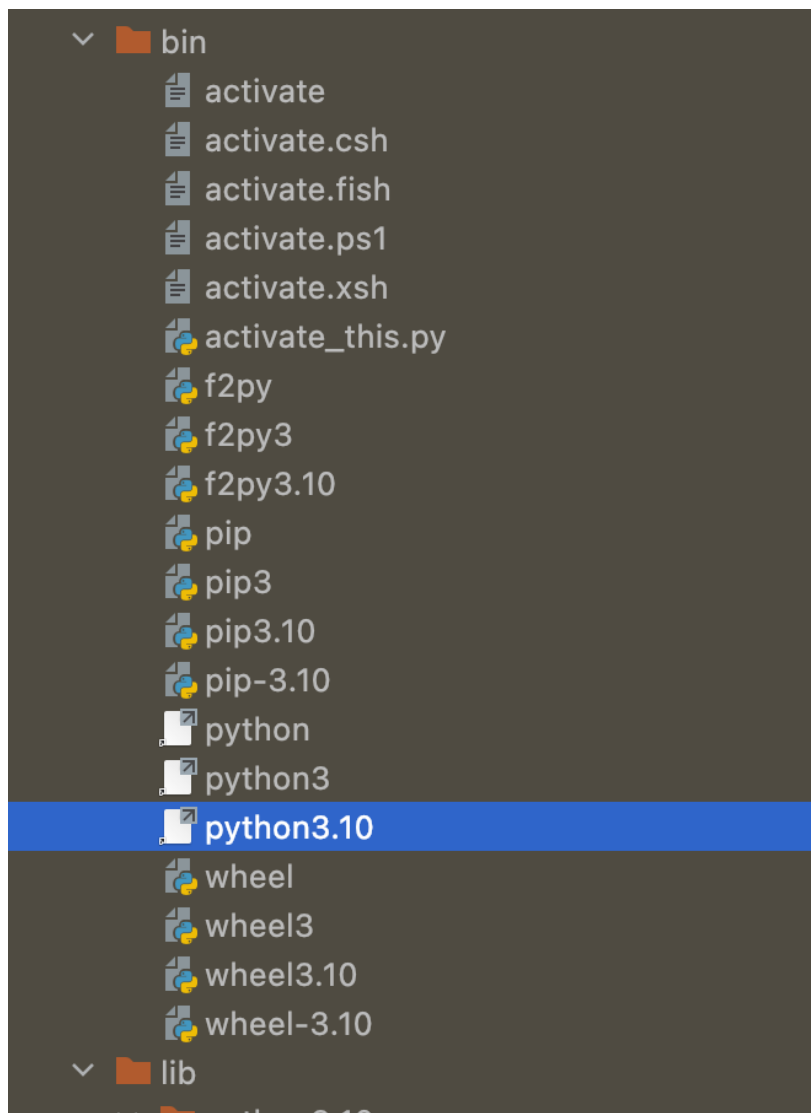
☐ Previously configured interpreter

Interpreter: ▼ ...

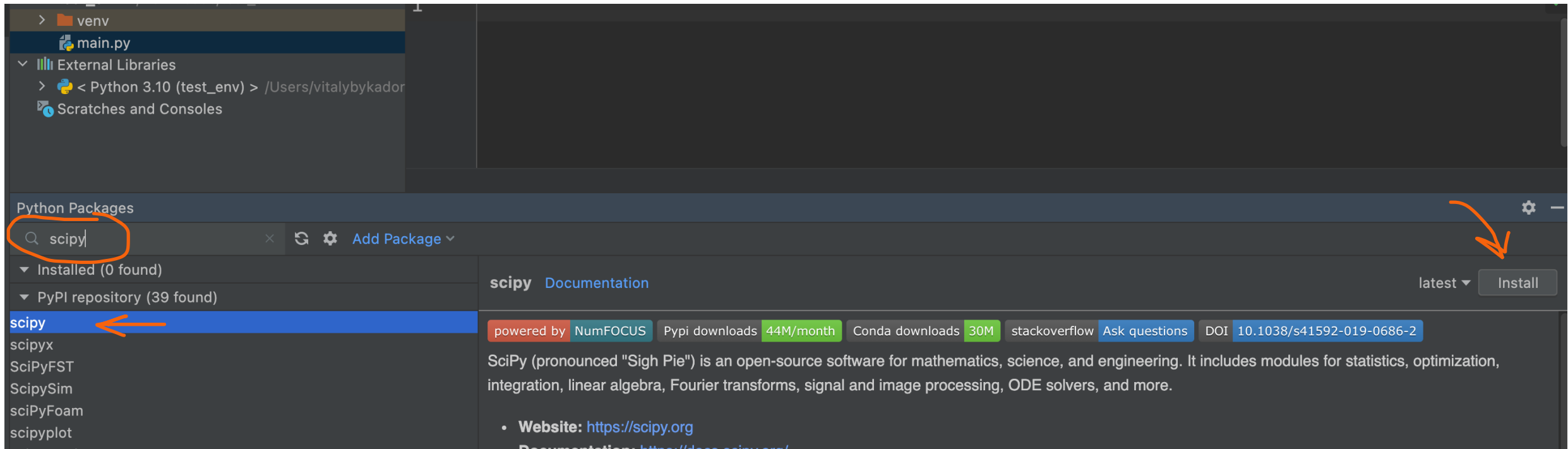
☐ Create a main.py welcome script

Create a Python script that provides an entry point to coding in PyCharm.

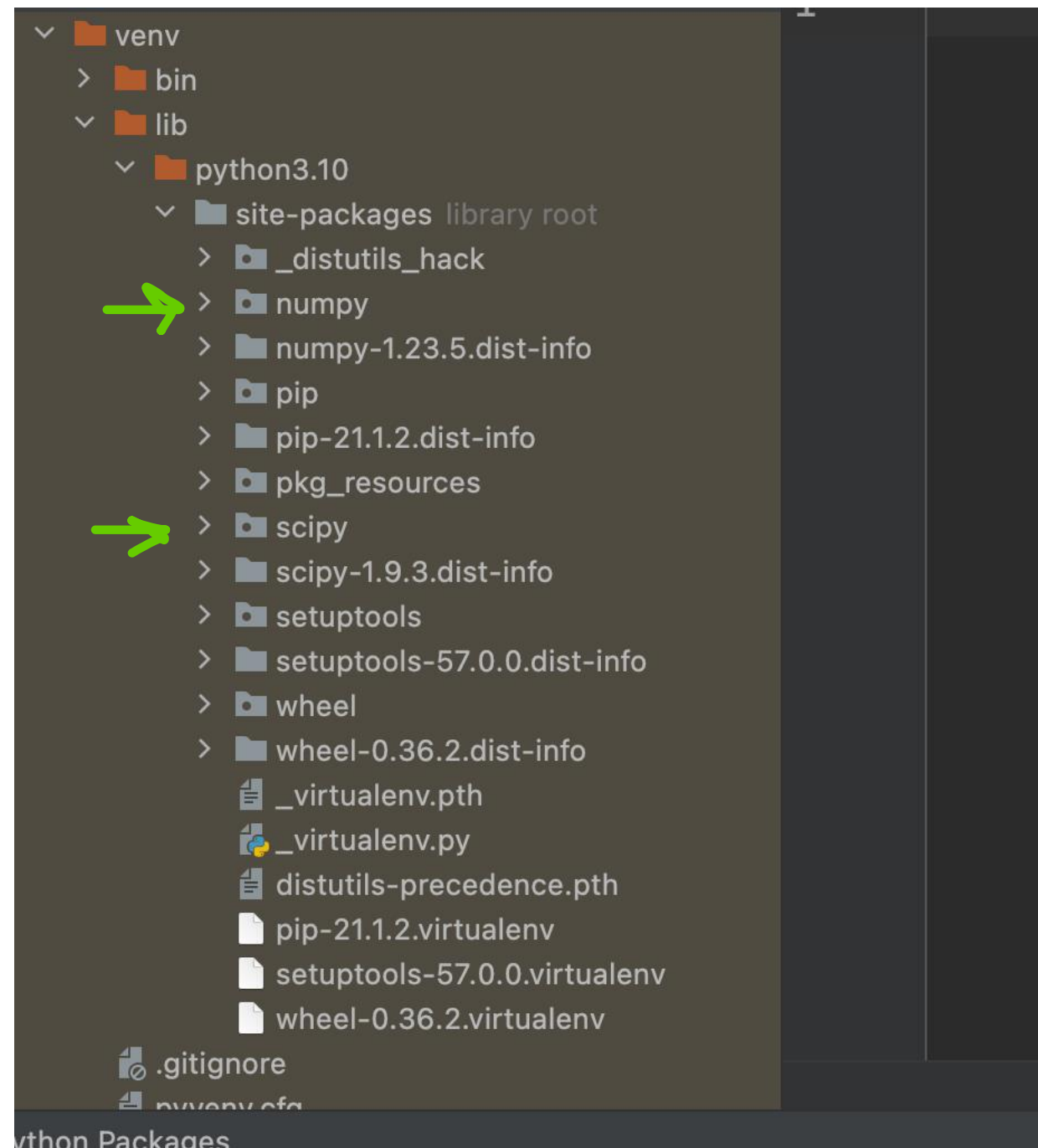
Виртуальная среда выполнения



Виртуальная среда выполнения

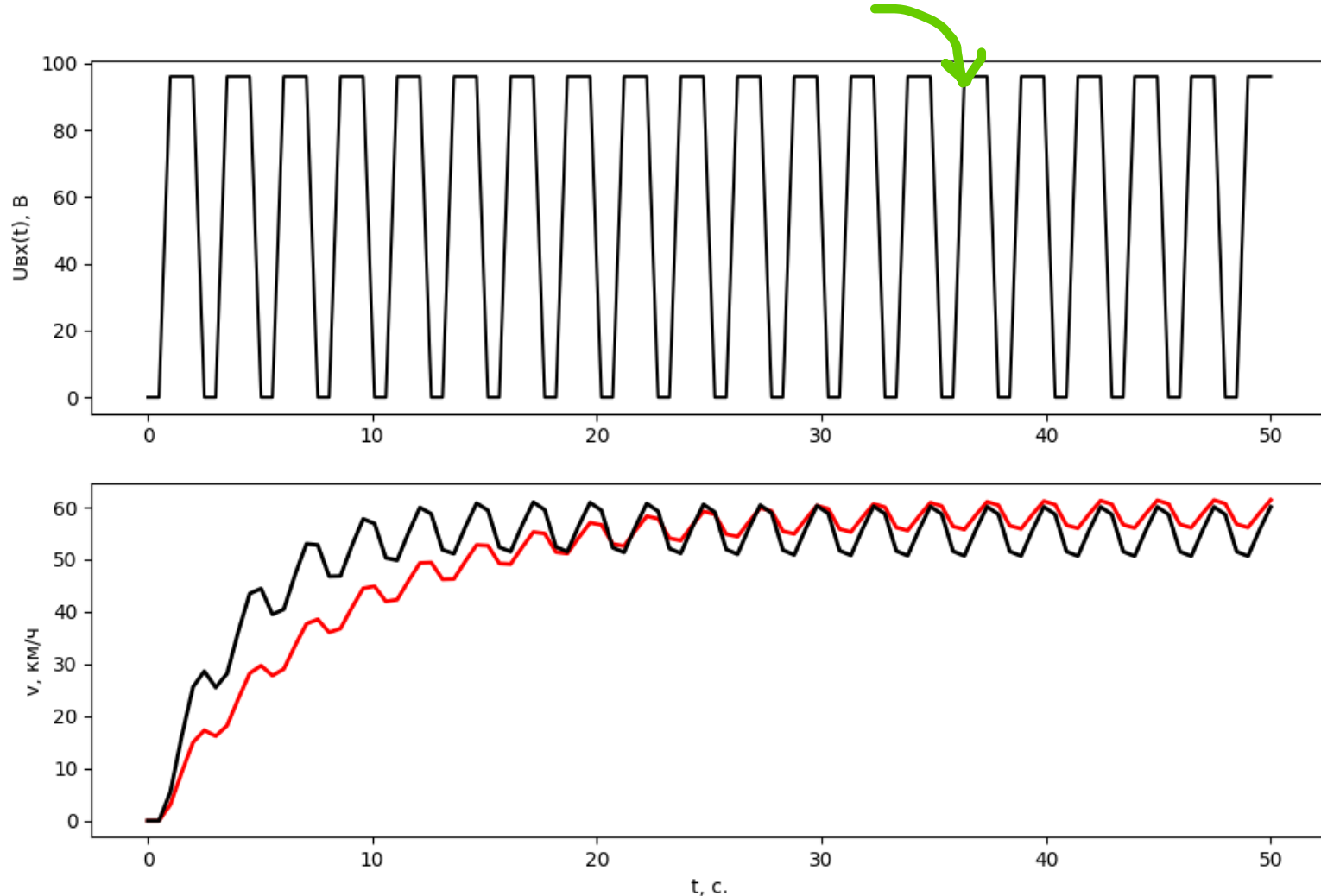


Виртуальная среда выполнения



Программный код на Python должен быть кодом на Python

Задача – нужна была импульсная последовательность



Программный код на Python должен быть кодом на Python

Решение № 01 в стиле языка
программирования Си.

```
6  def U_PWM(u: float, time: list) -> list:
7      result = []
8      i, j = 0, 0
9      while i < len(time):
10         if j == 0 or j == 1:
11             result.append(0)
12             j += 1
13             i += 1
14             continue
15         if j == 2 or j == 3:
16             result.append(u)
17             j += 1
18             i += 1
19         if j > 3:
20             j = 0
21     return result
```

16 строк

Программный код на Python должен быть кодом на Python

Решение № 02 в стиле языка
программирования Python.

```
7  def U_PWM(u_in: float, count_point_time: int) -> list[float]:  
8      i, result = 0, list()  
9      for _ in range(count_point_time):  
10         result.append(0 if i < 2 else u_in)  
11         i += -i if i > 3 else 1  
12     return result
```

6 строк