

# Введение и основные типы парадигм

Урок 1

Курс “Парадигмы программирования и языки парадигм”





## Имя и фамилия





Давайте знакомиться!

Немного о себе. Краткое описание  
в две-три строчки. Ключевая инф.

- ✨ Список проектов, которые делал(а);
- ✨ Или список компаний, с которыми работал(а);
- ✨ Может какие-то награды;
- ✨ Заслуги;
- ✨ И т.д.





## Типы заданий

-  4. Выявить отличия выбранной парадигмы в коде реализации
-  1. Кейсы: описание real-life задачи, ответом считается выбор парадигмы программирования или нескольких с аргументацией
-  2. Coding: есть задача, надо выбрать парадигму, написать код, чтобы работало и опять же аргументировать почему именно эта парадигма
-  3. Coding: псевдокод



## Цели семинара

-  Понять основные отличия между декларативной и императивной парадигмами
-  Начать решать задачи в рамках одной выбранной парадигмы



## План семинара



Викторина



Пишем код



Решаем кейсы



Подведение итогов



# Викторина





## Регламент

- 1 Прочитать код
- 2 Подумать в какой парадигме написана программа
- 3 Обсудить ответ



## Что за парадигма: максимум

```
1 def find_max(array: list) → int:
2     if len(array) > 0:
3         max_num = array[0]
4         for num in array:
5             if num > max_num:
6                 max_num = num
7     return max_num
```

Ответ: .. ?





## Что за парадигма: максимум

```
1 def find_max(array: list) → int:
2     if len(array) > 0:
3         max_num = array[0]
4         for num in array:
5             if num > max_num:
6                 max_num = num
7     return max_num
```

**Ответ:** Это *императивная* парадигма!

**Почему это так:** есть явное взаимодействие со списком и его элементами, например в 4 и 5 строках.



## Что за парадигма: максимум

```
1 def find_max(array: list) → int:
2     if len(array) > 0:
3         max_num = array[0]
4         for num in array:
5             if num > max_num:
6                 max_num = num
7     return max_num
```

### Декларативный стиль:

```
1 def find_max(array: list) → int:
2     return max(array)
```



## Что за парадигма: факториал

```
1 def calculate_factorial(n):  
2     if n == 0:  
3         return 1  
4     else:  
5         return n * calculate_factorial(n - 1)
```

Ответ: .. ?



## Что за парадигма: факториал

```
1 def calculate_factorial(n):  
2     if n == 0:  
3         return 1  
4     else:  
5         return n * calculate_factorial(n - 1)
```

**Ответ:** Это *декларативная* парадигма!

**Почему это так:** Рекурсивная функция вместо последовательности шагов для каждого числа



## Что за парадигма: факториал

```
1 def calculate_factorial(n):  
2     if n == 0:  
3         return 1  
4     else:  
5         return n * calculate_factorial(n - 1)
```

Как бы выглядел императивный стиль:

```
1 def calculate_factorial(n):  
2     factorial = 1  
3     for i in range(1, n + 1):  
4         factorial *= i  
5     return factorial
```



## Что за парадигма: простое число

```
1 def check_prime(number):  
2     if number < 2:  
3         return False  
4     for i in range(2, number):  
5         if number % i == 0:  
6             return False  
7     return True
```

Ответ: ... ?



## Что за парадигма: простое число

```
1 def check_prime(number):
2     if number < 2:
3         return False
4     for i in range(2, number):
5         if number % i == 0:
6             return False
7     return True
```

**Ответ:** Это *императивная* парадигма!

**Почему это так:** явное взаимодействие с циклом, мы получаем элемент на каждом шаге и проверяем его с помощью конкретных математических операций.



## Что за парадигма: простое число

```
1 def check_prime(number):
2     if number < 2:
3         return False
4     for i in range(2, number):
5         if number % i == 0:
6             return False
7     return True
```

Как выглядела бы декларативная парадигма:

```
1 def is_prime(number):
2     gen_list = [i for i in range(2, int(number ** 0.5) + 1)]
3     if number < 2:
4         return False
5     list_of_bool = list(map(lambda x: number % x != 0, gen_list))
6     return all(list_of_bool)
```





# Вопросы



Пишем код





## Регламент

- 1 Вместе читаем условия задачи
- 2 Вы самостоятельно решаете задачу
- 3 Вместе обсуждаем решение



# Поиск



## Поиск: императивный



### Контекст

Предположим, что нам хочется для любого массива чисел **array** и любого числа **target** узнать содержится ли **target** в **array**. Такую процедуру будем называть поиском.



### Задача

Реализовать **императивную** функцию поиска элементов на языке Python.



**Решение:** .. ?



## Поиск: императивный



### Контекст

Предположим, что нам хочется для любого массива чисел **array** и любого числа **target** узнать содержится ли **target** в **array**. Такую процедуру будем называть поиском.



### Задача

Реализовать **императивную** функцию поиска элементов на языке Python.



### Решение:

```
1 def search_imperative(array, target):
2     for num in array:
3         if num == target:
4             return True
5     return False
```



## Поиск: декларативный



### Контекст

Предположим, что нам хочется для любого массива чисел **array** и любого числа **target** узнать содержится ли **target** в **array**. Такую процедуру будем называть поиском.



### Задача

Реализовать **декларативную** функцию поиска элементов на языке Python.



**Решение:** .. ?



## Поиск: декларативный



### Контекст

Предположим, что нам хочется для любого массива чисел **array** и любого числа **target** узнать содержится ли **target** в **array**. Такую процедуру будем называть поиском.



### Задача

Реализовать **декларативную** функцию поиска элементов на языке Python.



### Решение:

```
1 def search_declarative(array, target):  
2     return target in array
```





# Доля чисел



## Доля чисел: императивный вариант

- **Условие**

На вход подается: список целых чисел **arr**

- **Задача**

Реализовать **императивную** функцию, которая возвращает три числа:

- Долю позитивных чисел
- Долю нулей
- Долю отрицательных чисел

- **Решение: .. ?**



## Доля чисел: императивный вариант

- **Условие**

На вход подается: список целых чисел **arr**

- **Задача**

Реализовать **императивную** функцию, которая возвращает три числа:

- Долю позитивных чисел
- Долю нулей
- Долю отрицательных чисел

- **Решение:**

```
1 def plus_minus(arr):
2     pos_cnt, neg_cnt, zero_cnt = 0, 0, 0
3     for el in arr:
4         if el > 0: pos_cnt += 1
5         elif el < 0: neg_cnt += 1
6         else: zero_cnt += 1
7     pos_frac = pos_cnt / len(arr)
8     neg_frac = neg_cnt / len(arr)
9     zero_frac = zero_cnt / len(arr)
10    return pos_frac, neg_frac, zero_frac
```



## Доля чисел: декларативный вариант

- **Условие**

На вход подается: список целых чисел **arr**

- **Задача**

Реализовать **декларативную** функцию, которая возвращает три числа:

- Долю позитивных чисел
- Долю нулей
- Долю отрицательных чисел

- **Решение: .. ?**



## Доля чисел: декларативный вариант

- **Условие**

На вход подается: список целых чисел **arr**

- **Задача**

Реализовать **декларативную** функцию, которая возвращает три числа:

- Долю позитивных чисел
- Долю нулей
- Долю отрицательных чисел

- **Решение:**

```
1 def plus_minus_decl(arr):
2     pos_cnt = len(list(filter(lambda x: x > 0, arr)))
3     neg_cnt = len(list(filter(lambda x: x < 0, arr)))
4     zer_cnt = len(list(filter(lambda x: x == 0, arr)))
5     counts = [pos_cnt, neg_cnt, zer_cnt]
6     return list(map(lambda count: count / len(arr), counts))
```



Решаем кейсы





## Регламент

- 1 Вместе читаем кейс
- 2 Вы думаете как можно его решить
- 3 Вместе обсуждаем решение



# Задача коммивояжера



## Кейс №1. Задача коммивояжера

### Контекст:

Предположим, вам заказали разработать программу для решения задачи коммивояжера с использованием генетического алгоритма.

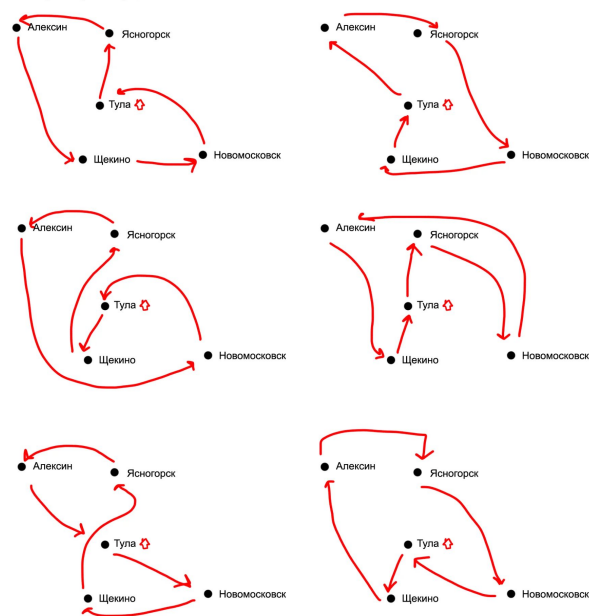
**Задача коммивояжера** - это задача поиска наиболее оптимального маршрута, проходящего через указанные вершины хотя бы по одному разу с последующим возвратом в вершину начала пути.

**ТЗ:** Ваша программа будет использована в реальном времени как микросервис. Ваша задача: разработать скрипт, протестировать, упаковать скрипт в микросервис и передать в продакшн.

**Задача:** Поразмышляйте. Какую парадигму будете использовать для разработки скрипта и почему именно её?



Некоторые варианты решения



## Кейс №1. Задача коммивояжера

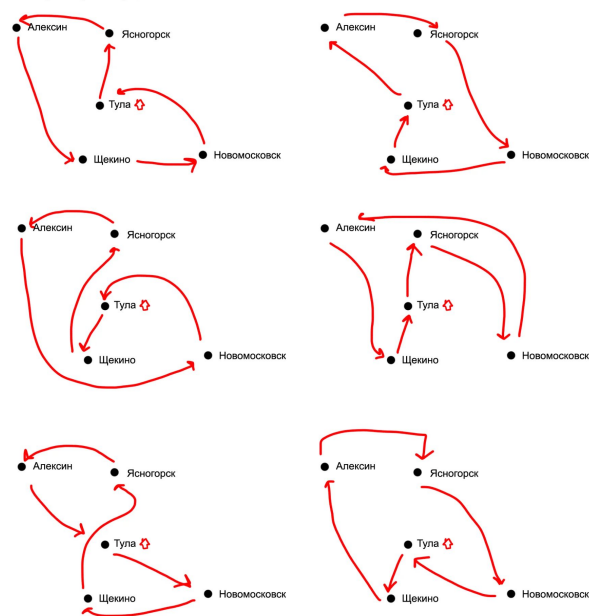
### Обсуждение:

Здесь все зависит от того какой алгоритм оптимизации вы хотите использовать для решения задачи:

- Если будете писать свой алгоритм с нуля - то почти наверняка стоит использовать императивную парадигму, для низкоуровневых манипуляций с переменными и памятью.
- Если хотите использовать готовый алгоритм из коробки (в Python как раз таких полно), то можно реализовать основную часть как декларативную. В таком случае, скорее всего у вас в программе будет какой-нибудь класс *Solver*, который на вход будет получать условия задачи и возвращать результат.



Некоторые варианты решения





# Визуализация

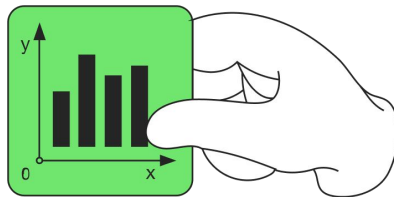
## Кейс №1. Визуализация

### Контекст:

На этот раз аналитики вам заказали скрипт для получения визуализации некоторых данных. Для разработки вы решили использовать Python-библиотеку “matplotlib”.

На вход подаются данные, на выходе какая-то аналитика. Таким образом, ваш скрипт - это последовательность преобразований данных + построение графиков.

**Задача:** Поразмышляйте. Какую парадигму будете использовать и почему?

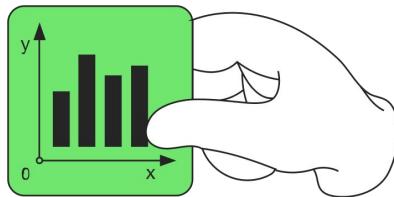


## Кейс №1. Визуализация

### Обсуждение:

Мы знаем, что мы должны использовать Python и конкретную библиотеку. Библиотека “matplotlib” - это удобный интерфейс для построения графиков, которая может быть использована как в императивном, так и в декларативном стиле.

Что мы знаем про скрипт. Поскольку на вход мы получаем данные, то большая часть программы будет написана в декларативном стиле, поскольку манипулировать данными в императивном стиле не очень удобно. С другой стороны, отдельные части кода можно оформить в императивном стиле, если это удобнее.





# Итоги семинара





## Итоги семинара



### Викторина “Что за парадигма”

- Решили 3 задачи на классификацию парадигм



### Пишем код

- Создали два решения (в двух парадигмах) для задачи поиска элемента
- Создали два решения (в двух парадигмах) для задачи “плюс-минус”



### Решаем кейсы

- Решили и обсудили кейс “Задача коммивояжера”
- Решили и обсудили кейс “Визуализация”



### Подвели итоги



# Домашнее задание







## Сортировка списка

Для разогрева на первое домашнее задание будет каноническая задача сортировки списка.



### Задача №1

Дан список целых чисел `numbers`. Необходимо написать в **императивном стиле** процедуру для сортировки числа в списке в порядке убывания. Можно использовать любой алгоритм сортировки.

```
1 def sort_list_imperative(numbers):  
2     # Императивный код здесь  
3     pass  
4     return numbers
```



### Задача №2

Написать точно такую же процедуру, но в **декларативном стиле**

```
1 def sort_list_declarative(numbers):  
2     # Декларативный код здесь  
3     pass
```



Конец семинара  
Спасибо за внимание!

