IT CAREER HUB

Урок 22.

Lambda-функции. Функции высшего порядка

Передача функций в качестве аргументов 2
Lambda-функции 5
Задания для закрепления 7
Парадигмы программирования 8
Функциональное программирование 10
Функции высшего порядка 11
Функции map, filter, reduce 12
Функция filtФункции высшего порядкаer 14
Функция reduce 16
Задания для закрепления 18
Функции any и all 20
Функция all 22
Задания для закрепления 24
Функция как ключ в sorted(), min(), max() 25
Параметр key в min() и max() 28
Практические задания 30

Передача функций в качестве аргументов

Функции в Python являются объектами, которые можно присваивать переменным, хранить в коллекциях и передавать в другие функции.

При передаче функции в качестве аргумента передаётся сама ссылка на функцию, а не её результат. Это означает, что функция не выполняется сразу, а только передаётся в другую функцию, где её можно вызвать.

Как это работает?

Когда функция передаётся в качестве аргумента:

- Не нужно указывать `()` при передаче, иначе передастся результат её выполнения.
- Функция может быть вызвана внутри другой функции по переданной ссылке с помощью `()`.

Пример 1: Передача функции без вызова

Python

```python
def square(x):
    return x * x


def cube(x):
    return x * x * x


def apply_function(func, value):
    return func(value)  # Вызываем переданную функцию внутри другой функции


result_square = apply_function(square, 5)  # Передаём функцию square без вызова (без скобок)
```

```
result_cube = apply_function(cube, 5)  # Передаём функцию cube без вызова (без скобок)

print(result_square)

print(result_cube)
```

Пример 2: Ошибка при передаче вызванной функции

Python

```
def square(x):

    return x * x



def apply_function(func, value):

    return func(value)



# Передается результат вызова функции, а не ссылка на функцию

result = apply_function(square(5), 5)  # Ошибка!
```

Пример 3: Хранение функций в коллекциях

Python

```
def add(x, y):

    return x + y



def multiply(x, y):

    return x * y
```

```
# Функции можно хранить в списках, словарях и передавать их динамически
operations = {
    "+": add,
    "*": multiply
}

choice = input("Выберите операцию (+, *): ")
# Из словаря получена функция и скобки с аргументами запускают её
print(operations[choice](10, 5))
```

Пример 4: Передача встроенной функции

Python

```
def process_data(func, data):
    return func(data)

# Можно передавать не только пользовательские функции, но и встроенные
result = process_data(abs, -10)
print(result)
```

IT CAREER HUB

Lambda-функции

Lambda-функция (или анонимная функция) — это небольшая, одноразовая функция, которая не требует явного объявления с `def`. Она используется для краткой записи простых операций и может быть передана как аргумент в другие функции.

Синтаксис:

```python
lambda arguments: expression
```

- `lambda` — ключевое слово для создания анонимной функции.
- `arguments` — параметры, которые принимает функция.
- `expression` — выражение, результат которого возвращается (без `return`).

Примеры:

Lambda с одним аргументом

```python
# Функция принимает число x и возвращает его квадрат

square = lambda x: x ** 2

print(square(4))

print(square(5))



# Аналог с def

def square(x):

    return x ** 2

print(square(4))

print(square(5))
```

Lambda с несколькими аргументами

Python

```python
# Функция принимает два аргумента и возвращает их сумму

add = lambda x, y: x + y

print(add(3, 5))

print(add(8, 9))



# Аналог с def

def add(x, y):

    return x + y

print(add(3, 5))

print(add(8, 9))
```

Lambda как аргументы других функций

Lambda-функции можно передавать как аргументы в другие функции, не создавая отдельные именованные функции.

Python

```python
def apply_func(func, numbers):

    return [func(num) for num in numbers]



result = apply_func(lambda x: x + 10, [5, 8, 3])

print(result)
```

Особенности lambda-функций

Lambda-функция всегда возвращает результат выражения. Lambda может содержать только одно выражение. Нет многострочных блоков кода (if, for и т. д.).

IT CAREER HUB

Задания для закрепления

Исправьте ошибку в коде

```python
operations = {

    "sum": lambda x, y: x + y,

    "mul": lambda x, y: x * y

}



print(operations("sum")(2, 3))

print(operations("mul")(2, 3))
```

Ответ: исправьте круглые скобки после `operations` на квадратные.

Объясните, что происходит при выполнении следующего кода. Произойдет ли ошибка?

```python
def add(x, y):

    return x + y



print((lambda f, a, b: f(a, b))(add, 3, 4))
```

Ответ: нет, код корректен.

Можно ли передавать встроенные функции Python в качестве аргументов другим функциям?

Ответ: Да, можно. Например, функции `abs`, `len`, `sum` и другие можно передавать как аргумент.

IT CAREER HUB

Парадигмы программирования

Парадигма программирования — это стиль написания и организации кода, который определяет способы решения задач в программировании. Различные парадигмы предлагают разные подходы к структурированию кода, обработке данных и управлению потоком выполнения программы.

Основные парадигмы программирования

Императивное программирование

- Описывает последовательность команд, которые изменяют состояние программы.
- Код состоит из инструкций, выполняемых шаг за шагом.
- **Пример языков:** Python, C, Java.

Процедурное программирование (подтип императивного)

- Код организован в функции (процедуры), каждая из которых выполняет определённую задачу.
- Используется разделение программы на логические блоки для повторного использования.
- **Пример:** Python, Pascal, C.

Объектно-ориентированное программирование (ООП)

- Основной концепцией является объект, который объединяет данные и методы для их обработки.
- Код организуется в классы и объекты, позволяя моделировать реальные сущности.
- **Пример:** Python, Java, C++.

Функциональное программирование

- Основано на функциях высшего порядка, чистых функциях и отсутствии изменения состояния.
- Предпочитает использование неизменяемых данных и рекурсии вместо циклов.
- **Пример:** Haskell, Lisp, Python.

Декларативное программирование

- Описывает **что** должно быть сделано, а не **как**.
- Включает функциональное программирование, а также языки разметки (SQL, HTML).
- **Пример:** SQL, Prolog, Haskell.

Современные языки программирования, такие как Python, поддерживают несколько парадигм, позволяя использовать гибридные подходы. Например, в Python можно писать как в процедурном стиле, так и использовать ООП и функциональные концепции. Выбор парадигмы зависит от задачи, удобства и требований проекта.

ICH IT CAREER HUB

Функциональное программирование

Функциональное программирование (Functional Programming) — это парадигма программирования, в которой основной единицей организации кода являются **функции**. Программы строятся из **чистых функций**, которые принимают аргументы и возвращают результат, не изменяя состояние программы.

Основные принципы функционального программирования

- **Чистые функции**
  - Функция всегда возвращает один и тот же результат при одинаковых входных данных.
  - Не изменяет внешние переменные (**отсутствуют побочные эффекты**).

- **Неизменяемость данных**
  - Данные **не изменяются**, вместо этого создаются **новые объекты**.
  - Изменяемость приводит к **непредсказуемому поведению**.

- **Функции как объекты**
  - Функции можно **передавать в другие функции**, **возвращать из функций**, **хранить в переменных**.
  - Это позволяет использовать **функции высшего порядка**.

- **Функции высшего порядка**
  - Функции, которые **принимают другие функции** в качестве аргументов **или возвращают их**.

- **Рекурсия вместо циклов**
  - Функциональные языки программирования часто используют **рекурсию** вместо циклов.

IT CAREER HUB

Функции высшего порядка

Функции высшего порядка — это функции, которые **могут принимать другие функции** в качестве аргументов и/или **возвращать функции** в качестве результата.

Признаки функций высшего порядка:

- **Принимают функции в качестве аргументов**
    - Функция передаётся как параметр и используется внутри.
- **Возвращают функции как результат**
    - Функция создаётся и возвращается другой функцией.

Часто используемые функции высшего порядка:

- **Встроенные функции высшего порядка**: `map()`, `filter()`, `reduce()`.
- **Использование `sorted()` с ключом сортировки**.

IT CAREER HUB

Функции map, filter, reduce

Функции **map**, **filter** и **reduce** — это функции высшего порядка, которые принимают другую функцию в качестве аргумента и применяют её к элементам переданного итерируемого объекта.

Функция map

Функция `map()` применяет переданную функцию к каждому элементу одного или нескольких итерируемых объектов и возвращает итератор с результатами.

Особенности:

- Используется для преобразования данных.
- Результат нужно преобразовать в список или другой итерируемый объект, чтобы увидеть значения.

Синтаксис:

```python
map(function, iterable)
```

- **`function`** — функция, применяемая к каждому элементу.
- **`iterable`** — итерируемый объект (список, кортеж, строка и т.д.).

Пример с одним объектом:

```python
numbers = [1, 2, 3, 4]

# Каждый элемент списка возводится в квадрат

squared = map(lambda x: x ** 2, numbers)

print(list(squared))  # [1, 4, 9, 16]
```

Пример с несколькими объектами:

```python
a = [1, 2, 3]

b = [4, 5, 6]

# Каждая пара элементов списков суммируется

result = map(lambda x, y: x + y, a, b)

print(list(result))  # [5, 7, 9]
```

Пример со встроенными функциями:

```python
group_numbers = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]

# К каждому кортежу применяется функция sum

result = map(sum, group_numbers)

print(list(result))  # [6, 15, 24]
```

IT CAREER HUB

Функция filter

Функция `filter()` используется для фильтрации элементов итерируемого объекта, возвращая только те, для которых переданная функция-предикат возвращает `True`.

Особенности:

- Используется для отбора нужных элементов, соответствующих условию.
- Если вместо функции передать `None`, будут выбраны только элементы, которые оцениваются как `True`.
- Результат нужно преобразовать в список или другой итерируемый объект, чтобы увидеть значения.

Синтаксис:

```python
filter(function, iterable)
```

- **`function`** — функция-предикат, определяющая условие для фильтрации.
- **`iterable`** — итерируемый объект.

Пример с функцией-предикатом:

```python
numbers = [1, 2, 4, 5, 7, 9, 10, 11]

# Из списка выбираются только чётные числа

even_numbers = filter(lambda x: x % 2 == 0, numbers)

print(list(even_numbers))  # [2, 4, 10]
```

Пример с `None`:

```python
data = [0, 1, False, True, '', 'Python', [], [1, 2, 3]]
```

```python
# Из списка выбираются только элементы, которые оцениваются как True

filtered_data = filter(None, data)

print(list(filtered_data))  # [1, True, 'Python', [1, 2, 3]]
```

IT CAREER HUB

Функция reduce

Функция `reduce()` последовательно применяет переданную функцию к элементам итерируемого объекта с накоплением результата, сводя его к одному значению. Она находится в модуле `functools`.

Особенности:

- Используется для агрегации данных (например, суммы или произведения).
- Работает с парой элементов, начиная с первых двух, и продолжает с результатом и каждым следующим значением.

Синтаксис:

```python
from functools import reduce

reduce(function, iterable, initializer)
```

- **`function`** — функция, принимающая два аргумента и возвращающая один результат.
- **`iterable`** — итерируемый объект.
- **`initializer`** (необязательно) — начальное значение для накопления результата.

Пример:

```python
from functools import reduce



numbers = [1, 2, 3, 4]

# Умножение всех элементов списка последовательно

result = reduce(lambda x, y: x * y, numbers)

print(result)  # 24
```

Пример с `initializer`:

```python
from functools import reduce



numbers = [1, 2, 3, 4]

# Умножение всех элементов списка, начиная с 10

result = reduce(lambda x, y: x * y, numbers, 10)

print(result)  # 240
```

Сравнение функций map, filter, reduce

Функция	Описание	Возвращает
`map`	Применяет функцию ко всем элементам итерируемого объекта	Итератор преобразованных элементов
`filter`	Фильтрует элементы на основе условия	Итератор отфильтрованных элементов
`reduce`	Последовательно применяет функцию, аккумулируя результат	Итоговое значение

IT CAREER HUB

Задания для закрепления

1. Какой результат будет выведен при выполнении следующего кода?

```python
words = ["apple", "banana", "cherry"]

lengths = map(len, words)

print(list(lengths))
```

Варианты ответов:

a) `[5, 6, 6]`
b) `[('apple', 5), ('banana', 6), ('cherry', 6)]`
c) `['apple', 'banana', 'cherry']`
d) **Ошибка**

✅ **Ответ: a**

2. Какой результат будет выведен при выполнении следующего кода?

```python
numbers = [5, 3, 4, 1, 5, 2]

filtered = filter(lambda x: x % 2 == 1, numbers)

print(list(filtered))
```

Варианты ответов:

a) `[5, 3, 1, 5]`
b) `[1, 3, 5, 5]`
c) `[1, 3, 5]`
d) `[True, True, False, True, True, False]`

✅ **Ответ: a**

3. Какой результат будет выведен при выполнении следующего кода?

Python

```python
from functools import reduce



numbers = [1, 2, 3, 4]

result = reduce(lambda x, y: x + y, numbers)

print(result)
```

Варианты ответов:

a) `[1, 2, 3, 4]`
b) `[1, 3, 5, 7]`
c) `[1, 3, 6, 10]`
d) `10`

✅ **Ответ: d**

IT CAREER HUB

Функции any и all

Ложные и истинные значения

Ложные значения: `0`, `None`, `False`, пустые коллекции (`[]`, `{}`, `()`, `set()`).
Истинные значения: Всё, что не считается ложным.

Функция any

Функция `any()` проверяет, содержит ли итерируемый объект хотя бы один истинный элемент.

Синтаксис:

```python
any(iterable)
```

- **`iterable`** — итерируемый объект (список, кортеж, строка, множество, словарь и т. д.), элементы которого нужно проверить.

Особенности:

- Возвращает `True`, если хотя бы один элемент в итерируемом объекте является истинным.
- Если все элементы ложные, возвращает `False`.
- Если объект пустой, возвращает `False`.
- Выполняет проверку поэлементно и останавливается, как только находит истинное значение (**ленивое вычисление**).

Пример использования:

```python
data = [0, None, False, 1]

print(any(data))  # True, так как есть хотя бы одно истинное значение (1)
```

```python
data = [0, None, False]

print(any(data))  # False, так как все элементы ложные



data = []

print(any(data))  # False, так как объект пустой
```

ICH IT CAREER HUB

Функция all

Функция `all()` проверяет, являются ли **все** элементы итерируемого объекта истинными.

Синтаксис:

```python
all(iterable)
```

- **`iterable`** — итерируемый объект, элементы которого нужно проверить.

Особенности:

- Возвращает `True`, если **все** элементы в итерируемом объекте истинные.
- Если хотя бы один элемент ложный, возвращается `False`.
- Возвращает `True` для **пустого объекта**, так как нет элементов, которые можно было бы считать ложными.
- Выполняет проверку поэлементно и останавливается, как только находит ложное значение.

Пример использования:

```python
data = [1, 2, 3]

print(all(data))  # True, так как все элементы истинные



data = [1, 0, 3]

print(all(data))  # False, так как 0 — ложное значение



data = []

print(all(data))  # True, так как объект пустой
```

IT CAREER HUB

Сравнение функций `any` и `all`

Характеристика	any	all
Результат	`True`, если хотя бы один элемент истинный.	`True`, если все элементы истинные.
Пустой объект	Возвращает `False`.	Возвращает `True`.
Остановка проверки	При нахождении первого истинного значения.	При нахождении первого ложного значения.
Применение	Проверка наличия истинных значений.	Проверка, что все значения истинные.

Пример использования:

```python
# Проверка, что хотя бы один объект соответствует условию

conditions = [x > 10 for x in [5, 20, 8]]

print(any(conditions))  # True



# Проверка, что все объекты соответствуют условию

conditions = [x > 0 for x in [5, 20, 8]]

print(all(conditions))  # True
```

IT CAREER HUB

Задания для закрепления

1. Что произойдёт, если передать в функцию `any` пустой список?

```python
data = []

print(any(data))
```

Варианты ответов:

a) Возвращается `True`
b) Возвращается `False`
c) Возникает ошибка
d) Возвращается `None`

✅ **Ответ: b**

2. Какой результат будет выведен при выполнении следующего кода?

```python
data = [1, 2, 3, "None"]

print(all(data))
```

Варианты ответов:

a) `True`
b) `False`
c) Ошибка
d) `None`

✅ **Ответ: a**

IT CAREER HUB

Функция как ключ в sorted(), min(), max()

В Python функции `sorted()`, `min()` и `max()` принимают необязательный параметр **`key`**, который определяет, по какому критерию выполнять сортировку или поиск минимального/максимального значения.

- В параметр `key` можно передавать не только встроенные функции (`len`, `abs`), но и **пользовательские функции** (созданные с помощью `def` или `lambda`).

Параметр `key` в `sorted()`

Функция `sorted(iterable, key=function)` сортирует элементы по значению, которое возвращает переданная функция.

Синтаксис:

```python
sorted(iterable, key=function, reverse=False)
```

Пример: Использование встроенной функции

```python
words = ['mango', 'grape', 'apple', 'strawberry', 'banana', 'pineapple', 'kiwi', 'blueberry']

# Сортировка по длине слов

result = sorted(words, key=len)

print(result)
```

Вывод:

```
['kiwi', 'apple', 'mango', 'grape', 'banana', 'orange', 'pineapple', 'strawberry']
```

Пример: Пользовательская функция

Python
```
def last_char_len(s):

    return s[-1], len(s)



words = ['mango', 'grape', 'apple', 'strawberry', 'banana', 'pineapple',
'kiwi', 'blueberry']

# Сортировка по последнему символу и длине слова

result = sorted(words, key=last_char_len)

print(result)
```

Пример: Анонимная функция `lambda`

Python
```
words = ['mango', 'grape', 'apple', 'Strawberry', 'Banana', 'pineapple',
'kiwi', 'blueberry']

# Сортировка по первому символу (игнорируя регистр) и по последнему символу

result = sorted(words, key=lambda x: (x[0].lower(), x[-1]))

print(result)
```

Пример: Сортировка списка кортежей

Python
```
students = [("Alice", 25), ("Bob", 20), ("Charlie", 23)]

# Сортировка списка кортежей по возрасту (второй элемент)

sorted_students = sorted(students, key=lambda x: x[1])

print(sorted_students)
```

Вывод:

```
Unset
[('Bob', 20), ('Charlie', 23), ('Alice', 25)]
```

IT CAREER HUB

Параметр key в min() и max()

Функции `min()` и `max()` с параметром `key` позволяют находить минимальный или максимальный элемент на основе вычисленного значения.

Синтаксис:

```python
min(iterable, key=function)

max(iterable, key=function)
```

Пример: Поиск самого длинного слова

```python
words = ["apple", "banana", "kiwi", "grapefruit"]

longest_word = max(words, key=len)

print(longest_word)
```

Вывод:

```
grapefruit
```

Пример: Поиск города с минимальным населением

```python
cities = [('New York', 8419600), ('Los Angeles', 3980400), ('Chicago', 2716000)]

smallest_city = min(cities, key=lambda x: x[1])

print(smallest_city)
```

Вывод:

```
('Chicago', 2716000)
```

IT CAREER HUB

Практические задания

1. Список квадратов чисел

Напишите функцию, которая сформирует список квадратов из полученного списка, без использования циклов или списковых включений.

Данные:

```python
numbers = [1, 2, 3, 4, 5]
```

Пример вывода:

```
[1, 4, 9, 16, 25]
```

Решение:

```python
def square_numbers(numbers):
    return list(map(lambda x: x ** 2, numbers))


numbers = [1, 2, 3, 4, 5]
print(square_numbers(numbers))
```

2. Сортировка по возрасту

Отсортируйте список кортежей (имя, возраст) по возрасту.

Данные:

```python
people = [
    ("Mike", 19), ("Nancy", 35), ("Charlie", 23), ("Oscar", 33), ("Eve", 29),
    ("Frank", 33), ("Bob", 20), ("Grace", 27), ("Isabella", 19), ("Jack", 24),
    ("Alice", 25), ("Kevin", 28), ("Laura", 31), ("Diana", 30), ("Henry", 19)
]
```

Пример вывода:

```
[('Mike', 19), ('Isabella', 19), ('Henry', 19), ('Bob', 20), ('Charlie', 23),
 ('Jack', 24), ('Alice', 25), ('Grace', 27), ('Kevin', 28), ('Eve', 29),
 ('Diana', 30), ('Laura', 31), ('Oscar', 33), ('Frank', 33), ('Nancy', 35)]
```

Решение:

```python
def sort_by_age(people):
    return sorted(people, key=lambda person: person[1])


print(sort_by_age(people))
```

3. Сортировка по возрасту и имени

Отсортируйте список кортежей (имя, возраст) по убыванию возраста,
в рамках одинакового возраста отсортируйте также по имени по алфавиту.

Данные:

```python
people = [
    ("Mike", 19), ("Nancy", 35), ("Charlie", 23), ("Oscar", 33), ("Eve", 29),
    ("Frank", 33), ("Bob", 20), ("Grace", 27), ("Isabella", 19), ("Jack", 24),
    ("Alice", 25), ("Kevin", 28), ("Laura", 31), ("Diana", 30), ("Henry", 19)
]
```

Пример вывода:

```
[('Nancy', 35), ('Frank', 33), ('Oscar', 33), ('Laura', 31), ('Diana', 30),
 ('Eve', 29), ('Kevin', 28), ('Grace', 27), ('Alice', 25), ('Jack', 24),
 ('Charlie', 23), ('Bob', 20), ('Henry', 19), ('Isabella', 19), ('Mike', 19)]
```

Решение:

```python
def sort_by_age_and_name(people):
    return sorted(people, key=lambda person: (-person[1], person[0]))


print(sort_by_age_and_name(people))
```

4. Данные о сотрудниках

У вас есть список сотрудников с их возрастами и зарплатами.
Напишите программу, которая:

- Фильтрует сотрудников старше 30 лет.
- Увеличивает зарплату отфильтрованных сотрудников на 20%.
- Возвращает обновлённый список сотрудников.

Данные:

```python
employees = [
    {"name": "Alice", "age": 25, "salary": 50000},
    {"name": "Bob", "age": 35, "salary": 60000},
    {"name": "Charlie", "age": 40, "salary": 70000},
    {"name": "Diana", "age": 28, "salary": 55000},
    {"name": "Eve", "age": 45, "salary": 80000}
]
```

Пример вывода:

```
[{'name': 'Bob', 'age': 35, 'salary': 72000.0},
 {'name': 'Charlie', 'age': 40, 'salary': 84000.0},
 {'name': 'Eve', 'age': 45, 'salary': 96000.0}]
```

Решение:

```python
def process_employees(employees):
    # Шаг 1: Фильтрация сотрудников старше 30 лет
    filtered = filter(lambda emp: emp["age"] > 30, employees)
    # Шаг 2: Увеличение зарплаты на 20%
    updated = map(lambda emp: {**emp, "salary": emp["salary"] * 1.2}, filtered)
    return list(updated)
```

```
result = process_employees(employees)

print(result)
```