2024-11-13

Элементы декларативного программирования в Python (окончание)

Корутины (сопрограммы)

Функции, выполнение которых многократно может быть остановлено, а затем продолжено с той же точки.

Корутины представляют собой более обобщенную форму функций (подпрограмм). Обычно у функций есть одна точка входа и одна точка выхода. У корутин может быть несколько точек входа, выхода и продолжения выполнения.

```
async def my_coroutine():
    print("Starting")
    await asyncio.sleep(1)
    print("Finished")

async def main():
    await my_coroutine()
    print("Main finished")

asyncio.run(main())
```

Note

В Python корутины изначально произошли от генераторов, т.к. генераторы по своему строгому определению ничем от них не отличаются.
До введения обязательного синтаксиса `async/await` в PEP 492 корутина могла выглядеть так:

```
import re

def grep(pattern):
    while True:
        line = yield
        if re.search(pattern, line):
            print(line)
```

```
search = grep("coroutine")
next(search)
search.send("Capybara")
search.send("Avocado")
search.send("Capybara eats an avocado and does coroutines")
```

После допускается только `async/await`, но под капотом в CPython они все еще мало чем отличаются.

```
import asyncio

async def grep(pattern):
   while True:
       line = await asyncio.to_thread(input)
       if re.search(pattern, line):
           print(line)

asyncio.run(grep("coroutine"))
```

Замыкания (Closures)

Функция, которая возвращает другую функцию с «замороженным» на момент определения набором данных

```
def multiplier(n):
    def mul(k):
        return n * k
    return mul

mul3 = multiplier(3) # функция умножения на 3

print(mul3(3), mul3(5))
# 9, 15
```

Частичное применение функции (partial)

Частичное применение функции предполагает на основе функции `N` переменных определение новой функции с меньшим числом переменных `M` < `N`, при этом остальные `N - M` переменных получают фиксированные «замороженные» значения.

```
from functools import partial

def mulPart(a, b):
    return a * b

par3 = partial(mulPart, 3)  # заморозка функции вокруг значения 3

print(par3(3), par3(5)) # частичное применение функции
# 9, 15
```

Функторы

Функтор — объект некоторого класса, который может быть вызван как функция.

В Python любой объект с определенным методом `__call__` является функтором.

```
class mulFunctor: # эквивалентный функтор
    def __init__(self, val1):
        self.val1 = val1
    def __call__(self, val2):
        return self.val1 * val2
fun3 = mulFunctor(3)

print(fun3(3), fun3(5)) # применение функтора
# 9, 15
```

Каррирование (Currying)

Каррирование (или транслитерацией карринг) — преобразование функции от многих переменных в функцию, принимающую 1 аргумент за 1 вызов (т.е. берущую их по одному).

Note

Такое преобразование было названо в честь Хаскелла Карри (того же, в честь кого назвали Haskell).
Впервые каррирование ввели М. Шейнфинкель и Г. Фреге

В некоторых «изначально» функциональных ЯП (Haskell, ML) существует отдельный оператор каррирования. В иных случаях любой язык, поддерживающий замыкания, включая Python, Perl, C++, поддерживает и каррирование.

```
def mul(x, y):
    return x*y

cur_mul = lambda x: lambda y: mul(x, y)

print(cur_mul(2)(3)) # 6

def mul2(x):
    return x*2
print(mul2(3))  # 6

# сравним с замыканием

def closure_mul(x):
    def mul(y):
        return x * y
    return mul

print(closure_mul(2)(3))  # 6
```

Визуализация концептов

The purest coding style, where bugs are near impossible - YouTube

https://www.youtube.com/watch?v=HlgG395PQWw

Практические задания 4. Обработка данных в декларативной парадигме

1. Загрузите список стран из countries.json
2. С помощью `map()` создайте новый список, изменив сделав название каждой страны прописным в списке стран.
3. С помощью `filter()`, чтобы отфильтровать страны, содержащие `'land'`.
4. С помощью `filter()`, чтобы отфильтровать страны, содержащие ровно шесть символов.
5. С помощью `filter()`, чтобы отфильтровать страны, содержащие шесть и более букв в списке стран.
6. С помощью `filter()` для отсеивания стран, начинающихся с буквы `'E'`.
7. С помощью `reduce()` объедините все страны и получите данное предложение на английском языке: Финляндия, Швеция, Дания, Норвегия и Исландия являются странами Северной Европы.
8. Решите предыдущие задачи, объединив две или более функций высшего порядка методов
9. Используя сначала каррирование, а затем замыкания, объявите функцию `categorize_countries()`, которая возвращает список стран с некоторым общим шаблоном (например, `'land', 'ia', 'island', 'stan'`), который можно менять.
10. Используя файл countries-data.json, выполните приведенные ниже задания в функциональной парадигме:
    1. Отсортировать страны:
        1. по названию,
        2. по столице,
        3. по численности населения
    2. Выявить произвольное число (начать с 10) наиболее распространенных языков и где их используют.
    3. Выявить произвольное число (начать с 10) наиболее населенных стран.

Лабораторная работа 4. Решение задачи коммивояжера в трехмерном пространстве муравьиным алгоритмом

1. Написать генератор случайных координат точек в трехмерном Евклидовом пространстве и «дорог» между ними, пути могут быть как однонаправленные, так и двунаправленные
2. Написать решение задачи коммивояжера для данных графов при помощи муравьиного алгоритма (Муравьиный алгоритм — Википедия, Муравьиный

алгоритм | Задача коммивояжёра / Хабр, Муравьиный алгоритм. Решение задачи коммивояжера / Хабр)

3. Визуализировать полученный путь при помощи `matplotlib` / `seaborn` / `Plotly` или иных библиотек визуализации
4. Проверить решение на наборах из 200, 500, 1000 точек

Что примерно должно получиться: