Семинар 11

Основы функционального программирования

Парадигмы программирования

- Императивное (процедурное)
- Декларативное
- Объектно-ориентированное
- Функциональное
- Другие..

Функциональное программирование

- Всё в программе построено на композиции взаимодействующих функций
- “Чистые” функции не зависят от внешнего состояния, имеют только вход и выход. Результат всегда однозначен.
- Все объекты относятся к неизменяемым (для новых данных используются новые объекты)

Итератор

Итератор - специальный объект, имеющий внутреннее состояние и встроенный метод ***__next__*** для перехода к следующему состоянию.

Если объект поддерживает преобразование в итератор, то можно пользоваться функцией iter(), например:

```
someObj = ([1, 2, 3], {})
for singleObj in iter(someObj):
    print(singleObj)
```

Итератор

```
# Пример итератора, возвращающего квадраты первых 10 чисел
# именованные изменяемые параметры в функции ведут себя, как "статические" данные
def get_squares(innerValue=[0]):
    innerValue[0] += 1  # "сохраняем" новое значение
    if innerValue[0] < 11:
        return innerValue[0] ** 2
    # None возвращается "по умолчанию", дополнительно ничего писать не нужно
# первый параметр - функция (обязательно без параметров),
# второй - возращаемое значение, которое является условием выхода
testIt = iter(get_squares, None)
# главное свойство - отложенное выполнение. Данные доставляются только по мере необходимости
print(*testIt, end=' ')
```

* - Распаковка

“*” как оператор распаковки применяется к параметрам функции.
При этом происходит следующее: из объекта, к которому применяется распаковка, извлекаются отдельные элементы и передаются в качестве отдельных параметров.

```
def func1():
    return 2, 5, 8

def func2(x, y, z):
    print(x + y + z)

func2(*func1())
```

```
# Что будет напечатано?
str1 = "1,2,3,4,5,6,7,8"
print(max(*map(int, str1.split(','))))
```

Генератор

Генератор - специальная функция, которая ***не*** завершается с оператором *return*. Вместо этого используется оператор *yield,* по которому происходит временный выход из функции, при этом все необходимые данные сохраняются.

```
def generate_squares():
    for i in (1, 2, 3):
        yield i * i

generator = generate_squares()
print("First Element", generator.__next__())
print("Others:", end=" ")
for i in generator:
    print(i, end=" ")
# print("Element:, generator.__next__()) <-- exception, генератор закончился,
next уже некуда
```

Задача 1. Генератор чисел Фибоначчи

Создать объект генератор, возвращающий очередное число Фибоначчи при обращении. Максимальное число - n.

```
def fibonacci(n):
    fib1, fib2 = 0, 1
    for i in range(n):
        fib1, fib2 = fib2, fib1 + fib2
        yield fib1


for fib in fibonacci(20):
    print(fib, end=' ')
print()

print('Сумма первых 100 чисел Фибоначчи равна', sum(fibonacci(100)))
print(list(fibonacci(16)))
print([x * x for x in fibonacci(14)])
```

Функции map и zip

`map(func, iterator1, [iterator2, …])`

`func` - функция, которую нужно применить к элементам последовательности(тям)
`iterator` - последовательность (или iterable-объект)

```
words = ["dog", "frog", "cat", "tiger"]
print(list(map(len, words)))    # [3, 4, 3, 5]


print(list(map(lambda x, y: x * y, [1, 2], [3, 4, 5]))) # [3, 8]
```

`zip(iterator1, [iterator2, …])`

```
a = [1, 2]
b = [3, 4]
print(list(zip(a, b))) # [(1, 3), (2, 4)]
```

Функции filter и enumerate

filter(func, iterator1)

func - функция, которую нужно применить к элементам последовательности
iterator - последовательность (или iterable-объект)
Возвращает последовательность элементов из исходной, для которой func возвращает **True**

```
myList = [1, 2, 3, 4, 5]
print(*filter(lambda x: x % 2, myList))
```

enumerate(iterator1[, start=0])

```
for item in enumerate(myList, 42):
    print(item, end=" ")  # (42, 1) (43, 2) (44, 3) (45, 4) (46, 5)
```

Задача 2.

Без использования циклов преобразовать список чисел в строку, записывая числа через запятую.

```
numbersList = list(map(int, input().split()))
str1 = ", ".join(list(map(str, numbersList)))
print("Res = %s" % str1)
```

Задача 3. Распределение полных квадратов

Без применения циклов сгенерировать список из 100 случайных чисел в диапазоне от 0 до 100. Вывести сколько процентов из них являются полными квадратами.

```
import math
import random


length = 100
lst = [random.randrange(1, 100) for x in range(length)]
numberOfSquares = len(list(filter(lambda x: math.sqrt(x).is_integer(), lst)))
percentOfSquares = round(numberOfSquares / len(lst) * 100)
print("Процент полных квадратов в последновательности равен %d" % percentOfSquares)


# тоже самое в 1 строку
percentOfSquares = round(len(list(filter(lambda x: math.sqrt(x).is_integer(),
[random.randrange(1,100) for x in range(length)]))) / length * 100)
print("Процент полных квадратов в последновательности равен %d" % percentOfSquares)
```