

Занятие 15

Рекурсия – мемоизация

Регулярные выражения

Объясните, что делает каждая строка

```
a = [i for i in range(10)]  
b = (i for i in range(10))  
c = {i for i in range(10)}  
d = {x: x**2 for x in range(10)}
```

В частности, что напечатает каждый `print(a, b, c, d)`

Что напечатает:

```
print(print(10))
```

Задача 14-1

- Напишите рекурсивную функцию, которая вычисляет количество цифр введенного натурального числа

Задача 14-2

- Напишите рекурсивную функцию, которая вычисляет сумму цифр натурального числа

Задача 14-3

- Напишите рекурсивную функцию `tri_2(n)`, которая печатает два треугольника.
- Например, для $n = 5$:

- *****
- *****
- ***
- **
- *
- **
- ***
- ****
- *****

- Подсказка: одна строка печатается до вызова функции, а вторая после вызова

Генерация исключений в Python

- Для принудительной генерации исключения используется инструкция **raise**.
- try:
- **raise** Exception("Что то пошло не так")
- except Exception as e:
- print("Message:" + str(e))

Выполните этот код

Пользовательские исключения

- В Python можно создавать собственные исключения.
- Такая практика позволяет увеличить гибкость процесса обработки ошибок в рамках той предметной области, для которой написана ваша программа.
- ```
class NameTooShortError (ValueError) :
 pass

def validate(name) :
 if len(name) < 10 :
 raise NameTooShortError
```
- Выполните этот код

# Пользовательские исключения в Python

- Для реализации собственного типа исключения необходимо создать класс, являющийся наследником от одного из классов исключений.
- `class NegValException(Exception):`
- `pass`
- `try:`
- `val = int(input("input positive number: "))`
- `if val < 0:`
- `raise NegValException("Neg val: " + str(val))`
- `print(val + 10)`
- `except NegValException as e:`
- `print(e)`



# yield from <iterable>

```
def fun_gen(n):
 for x in range(n):
 yield x
```

# Можно упростить

```
def fun_gen(n):
 yield from range(n)
```

# Например:

```
yield from 'abcdef'
```

```
yield from [11, 22, 33, 'abc', {1:111}]
```

```
yield from {1,22,333,4444}
```

# В каком порядке напечатаются эти числа?

# Вложенные генераторы

```
def fun_gen1():
 yield "Красный"
 yield "Зеленый"
 yield "Синий"

def fun_gen2():
 yield "Круглый"
 yield from fun_gen1()
 yield "Квадратный"

print(*fun_gen2())
```

# Что напечатает эта программа?

# Конвейеры генераторов

```
def integers(n):
 for i in range(1, n + 1):
 yield i

def evens(iterable):
 for i in iterable:
 if not i % 2:
 yield i

def squared(iterable):
 for k in iterable:
 yield k * k

chain = squared(evens(integers(10)))
print(*chain) # Что будет напечатано?
```



# Рекурсия – расчет факториала

```
def fact(n):
```

```
 if n == 1:
```

```
 return 1
```

# **базовый** случай – вариант решения

# без рекурсии

```
 else:
```

```
 return n * fact(n - 1)
```

# **рекурсивный** случай – сведение задачи

# к более простой

```
print(fact(1))
```

```
print(fact(2))
```

```
print(fact(3))
```

# Мемоизация

- Напишите рекурсивную функцию расчета чисел Фибоначчи:
- 0, 1, 1, 2, 3, 5, 8, 13, и т.д.

```
d = {}
def fibo(n):
 d[n] = d.get(n, 0) + 1 # Посчитаем сколько раз вызывалась функция при
 # каждом значении n
 if n == 1: return 0
 elif n == 2: return 1
 else: return fibo(n - 1) + fibo(n - 2)
fibo(15)
print(d)
{15: 1, 14: 1, 13: 2, 12: 3, 11: 5, 10: 8, 9: 13, 8: 21, 7: 34, 6: 55, 5: 89, 4: 144, 3: 233, 2:
377, 1: 233}
```

# Что если хранить уже вычисленные результаты?

```
d = {}
res = {}
def fibo(n):
 d[n] = d.get(n, 0) + 1
 if n not in res: # если ключ n уже есть в словаре значений, то просто возвращаем его
 if n == 1: return 0
 elif n == 2: return 1
 else:
 res[n] = fibo(n - 1) + fibo(n - 2) # сначала запоминаем результат и потом возвращаем
 return res[n]
fibo(15)
print(res)
print(d)
res = {3: 1, 4: 2, 5: 3, 6: 5, 7: 8, 8: 13, 9: 21, 10: 34, 11: 55, 12: 89, 13: 144, 14: 233, 15: 377}
d = {15: 1, 14: 1, 13: 2, 12: 2, 11: 2, 10: 2, 9: 2, 8: 2, 7: 2, 6: 2, 5: 2, 4: 2, 3: 2, 2: 2, 1: 1}
```

# match ... case...

- `def fibo(n):`
- `match n:`
- `case 1: return 1`
- `case 2: return 1`
- `case _: return fibo(n-1) + fibo(n-2)`
- `print(fibo(int(input())))`



# Задание

Создайте рекурсивную функцию, которая получает как аргумент список, который может содержать списки, которые могут содержать списки и т.д.

Например: [1, 2, [11, 22, [111, 222, [1111, 2222, 3333], 333, 444, [555, 666]], 3], 4]

Функция должна составить список, состоящий только из чисел, например:

[1, 2, 11, 22, 111, 222, 1111, 2222, 3333, 333, 444, 555, 666, 3, 4]

# Настройка глубины рекурсии

```
import sys
```

```
sys.getrecursionlimit()
```

```
sys.getrecursionlimit() = ... # Установка лимита
```



# RE

Regular expressions

# re – модуль Питона (import re)

- Который позволяет использовать все богатство регулярных выражений по поиску, замене, выборкам текстов
- Используется не только в Питоне
- Почему недостаточно большого количества функций по работе со строками: поиск, замена, вставка, удаление подстрок?
- Есть классы задач по обработке символьной информации:
  - Проверить текст на соответствие шаблону, например (адрес электронной почты), поиск по шаблону и др.

# Регулярное выражение

- Регулярное выражение — это строка, задающая шаблон поиска подстрок в тексте.
- Одному шаблону может соответствовать много разных строк.
- Регулярное выражение состоит из обычных символов и специальных командных последовательностей.
- Например, `\d` задаёт любую цифру, а `\d+` — задает любую последовательность из одной или более цифр., замены или разделения

# re.findall()

```
re.findall(pattern, string, flags=0)
```

```
import re
```

```
string = "Числа 99, 72, 81 и 999 делятся на 9"
```

```
re.findall(r"9", string)
```

```
99, 999, 81, [8, 9], [7, 8, 9], [7-9], [0-9]
```

```
\d, \d\d, \d\d\d, \d{3}, \d{1, 3}
```

```
\d+ \d* r" \d*"
```

```
\d{2}, r"\d{2}," r" \d{2}" r"\d, \d"
```

# Примеры

| Регулярное выражение | Значение                                                            |
|----------------------|---------------------------------------------------------------------|
| ITMO                 | В точности “ITMO”                                                   |
| \d{5}                | 5 цифр подряд, например: ‘12345’ или ‘88888’                        |
| \d\d/\d\d/\d\d\d\d   | Дата в формате ДД/ММ/ГГГГ, например 12/04/1961 или 35/78/9876       |
| \b\w{3}\b            | Слово точно из трех букв                                            |
| [+-]?\d+             | Целое число со знаком или без знака. Хотя бы одна цифра должна быть |



# Использование регулярных выражений

- Проверка на соответствие фрагментов текста некоторым критериям
- Поиск подстрок по заданному критерию для того, чтобы что-то полезное сделать с ними
- Замена подстрок, удовлетворяющих шаблону на другие подстроки

# Сырые (raw) строки

В сырой строке отключается экранирование.

Это значит, что обратная косая черта считается самостоятельным символом.

Основное применение сырых строк – работа с регулярными выражениями.

```
print("\\\\")
```

```
print('\\')
```

```
print(r"\\\\")
```

```
print(r'\\')
```

# Спецсимволы

. ^ \$ \* + ? { } [ ] \ | ( ) – для их написания их необходимо экранировать, т.е. поставить перед ними знак \

Шаблон регулярного выражение состоит из символов двух видов: обычных символов и метасимволов

# Диапазоны

- Регулярное выражение `[0-9]` функционально эквивалентно выражению `[0123456789]`.
- `[A-Z]` соответствует всем символам латинского алфавита верхнего регистра от A до Z
- `[a-z]` соответствует всем символам латинского алфавита нижнего регистра от a до z
- `[A-F]` соответствует всем символам латинского алфавита верхнего регистра от A до F
- `[А-Я]` соответствует всем символам русского алфавита верхнего регистра от А до Я
- `[а-ф]` соответствует всем символам русского алфавита нижнего регистра от а до ф

# Немного метасимволов

\d – любая цифра

\w – любая буква

\b – начало и конец слова

+ – один или больше элементов

\* – ноль или больше элементов

? – ноль или один элемент

Что найдет следующее выражение?

```
import re
```

```
string = "0abracadabra1"
```

```
regex = r".a." # a если regex = r"\da\w" regex = r"a\d"
```

```
re.findall(regex, string)
```

# Задание

- Дополните приведенный ниже код, чтобы переменная `regex` содержала регулярное выражение, которому соответствуют последовательности цифр, соответствующие числам от 100 до 199 включительно

```
import re
```

```
regex = r"?????" # Какой паттерн нужно задать, чтобы найти все числа,
которые попадают в интервал от 100 до 199?
```

```
re.findall(regex, string)
```

Подсказка: нужно задать три цифры. `\d` соответствует любой цифре.

# Задание

```
string = "Косой косой косил траву на косе"
import re
regex = r"?????" # Какой паттерн нужно задать, чтобы найти все слова?
re.findall(regex, string)

Подсказка: начало и конец слова это \b
буквы обозначаются \w
несколько букв \w+

Какой паттерн нужно задать, чтобы найти все слова, где есть три буквы "кос"
```

# Что нужно сделать, чтобы попали и косой и Косой

Использовать | - он используется как логическое ИЛИ, если нужно использовать любой из нескольких паттернов

Например:

```
regex = r"\b\w*кoc\w*\b|\b\w*Кoc\w*\b"
```

или

```
regex = r"\b\w*[Кк]oc\w*\b"
```



# Примеры

| Выражение | Примеры              |
|-----------|----------------------|
| c.t       | cat, cut, c#t, c{t   |
| c[aui]t   | cat, cut, cit        |
| c[a-c]t   | cat, cbt, cct        |
| c[^aui]t  | cbt, cct, c2t, и др. |
| c[^a-c]t  | cdt, cet, c%t        |

Знак ^ в квадратных скобках означает КРОМЕ

Давайте потренируемся.

```
string = "cat cet cit cot cut c#t c{t c2t"
```

Пробуйте разные regex из левого столбика на этой строке

# Наиболее употребительные метасимволы

| Use    | To match any character                    |
|--------|-------------------------------------------|
| [set]  | Один из этих символов                     |
| [^set] | Ни один из этих символов                  |
| [a-z]  | Любой из символов от a до z               |
| [^a-z] | Ни один из символов от a до z             |
| .      | Любой из символов кроме \n (новая строка) |
| \char  | Экранирование спецсимвола                 |

# Некоторые полезные символы

| Метасимвол | Диапазон              | Описание                                          |
|------------|-----------------------|---------------------------------------------------|
| \d         | [0-9]                 | любая цифра                                       |
| \D         | [^0-9]                | любой нецифровой символ                           |
| \w         | [0-9a-zA-Zа-яА-ЯёЁ_]  | любой алфавитно-цифровой символ и символ _        |
| \W         | [^0-9a-zA-Zа-яА-ЯёЁ_] | любой символ, отличный от алфавитно-цифрового и _ |
| \s         | [ \f\n\r\t\v]         | любой пробельный символ                           |
| \S         | [^ \f\n\r\t\v]        | любой непробельный символ                         |
|            |                       |                                                   |

# Установление соответствия в случае интервала-диапазона

| $\backslash d\{1,3\}$ | От 1 до 3 цифр     |
|-----------------------|--------------------|
| $\backslash w\{2\}$   | Ровно две буквы    |
| $\backslash d\{2,\}$  | Две и больше цифры |
| $\backslash w\{,5\}$  | Не больше 5 букв   |
|                       |                    |
|                       |                    |

# Задачи – напишите и проверьте regex, соответствующий...

- Последовательностям формата xxx.xxx , где x – любой символ.
- Телефонным номерам формата xxx-xxx-xxxx, где x – произвольная цифра, например: 921-123-4567
- Автомобильным номерам. Допустим, что W - любые заглавные латинские буквы, D – любые цифры, формат: WDDDWDD или WDDDWDDD, например: A123BC78 или A123BC178

# re.sub()

```
import re
```

```
text = 'Java самый популярный язык программирования в 2023
году.'
```

```
res = re.sub(r'Java', r'Python', text)
```

```
print(res)
```

# re.subn()

```
import re
```

```
text = 'http://www.lksjflskj.ru'
```

```
res, n = re.subn(r'Java', r'Python', text)
```

```
print(res, n)
```

# Задание

Дан текст, в котором есть телефонные номера, начинающиеся с (095). Замените этот код на (812) во всем тексте.

Внимание! 095 без скобок заменять не надо !!!

Подсказка. Используйте функцию `re.sub()`



# Задача 15-1

Создайте рекурсивную функцию, которая получает как аргумент dct словарь, который может содержать словари, которые могут содержать словари и т.д. и как аргумент x значение ключа.

Например:

```
dct = {1:1, 2:2, 3:{2:22, 3:{1:111, 2:222, 3:{0:1111, 1:2222, 2:3333}, 1:333}, 1:11},},
6:22}
```

```
x = 1
```

Функция должна составить список, состоящий только из значений словаря с ключем x.

Например:

```
[1, 111, 2222, 333, 11]
```

## Задача 15-2

Напишите функцию, которая принимает строку символов, и печатает все содержащиеся в ней номера автомашин по следующему правилу:

LDDDLL78 или LDDDLL178,

где L – буквы, совпадающие по начертанию в русском и латинском алфавите, D – цифры от 0 до 9.

Например, A123BC78 или X666XX178

# Задача 15-3

Напишите функцию, которая находит в строке все телефонные номера, которые удовлетворяют следующим шаблонам:

+7(812)DDD-DDDD, +7(812)DDD-DD-DD, +7(921)DDD-DDDD, +7(921)DDD-DD-DD

где D любая цифра