

## Тема 1.

# Введение в язык программирования Python. Алгоритмические средства языка Python

продолжение

# Регулярные выражения в Python

---

# Регулярные выражения в Python

---

- В Python для работы с регулярными выражениями предусмотрен модель **re**.
- И шаблоны и искомые тексты могут быть представлены как в Unicode, так и в ANSI. Однако смешивать их в одном операторе нельзя!

# Регулярные выражения в Python

---

- **Чаще всего регулярные выражения используются для:**
  - **поиска в строке**
  - **разбиения строки на подстроки**
  - **замены части строки**

# Регулярные выражения в Python

---

- Как правило, регулярные выражения содержат символ **'\'** для обозначения специальных последовательностей и для экранирования символов
- Однако в Python символ **'\'** используется для этих же целей в строках.
- В результате для поиска **'\'** нужно использовать шаблон **'\\\\'**.

# Регулярные выражения в Python

---

- Кроме того, в настоящее время некорректные escape-последовательности в строках Python теперь генерирую предупреждение `DeprecationWarning`, а в будущем оно станет `SyntaxError`.
- *Решение*: использовать raw-строки для написания регулярных выражений
- Тогда шаблон для поиска символа `\` `r'\'`.

# Регулярные выражения в Python

---

- Синтаксис регулярных выражений стандартный.
- Регулярное выражение включает:
  - набор возможных символов
  - квантификатор

# Регулярные выражения в Python

---

- **Наборы возможных символов:**
  - `.` `^` `$` `\` `|`
  - `\w` `\W` `\d` `\D` `\s` `\S` `\b` `\B` `\A` `\Z`
  - `[...]`, `[^...]`, `(...)`, `(?...)`
  - и др.
- **Квантификаторы (жадные, ленивые):**
  - `*`, `?`, `+`, `*?`, `??`, `+``?`
  - `{m,n}`, `{,m}`, `{n,}`, `{m,n}?`, `{,m}?`, `{n,}?`



# Регулярные выражения в Python

---

- Некоторые нововведения в regex Python

- **(?P<name> ...)** – именованная группа

Пример:

Найти в тексте часть, заключенную в кавычки  
(одинарные или двойные)

**(?P<quote>["']).\*?(?P=quote)**

The diagram illustrates the components of the regular expression `(?P<quote>["']).*?(?P=quote)` using colored brackets:

- A green bracket under `(?P<quote>` and a blue bracket under `["']` are connected by a red bracket, representing the named group `(?P<quote>["'])`.
- A red bracket under `.*` and a red bracket under `?` are connected by a red bracket, representing the quantifier `.*?`.
- A red bracket under `(?P=quote)` represents the second named group.

# Регулярные выражения в Python

---

- **Наиболее часто используемые методы:**
  - **`re.match(pattern, string)`**
  - **`re.search()`**
  - **`re.findall()`**
  - **`re.split()`**
  - **`re.sub()`**
  - **`re.compile()`**

# Регулярные выражения в Python

---

- **re.match(pattern, string)** – метод ищет соответствие по заданному шаблону в начале строки. Если совпадения с шаблоном нет, то метод вернет **None**, иначе **Match-объект**.

```
s = "Python language"
```

```
print(re.match("Python", s))
```

```
<re.Match object; span=(0, 6), match='Python'>
```

```
print(re.match("lang", s))
```

```
None
```

# Регулярные выражения в Python

---

- `re.start()` и `re.end()` – методы возвращают начальную и конечную позиции найденной строки.

```
import re
s = "Python language"
res=re.match(r"\w+", s)
if res:
    print(res.start())
    print(res.end())
else: print("Строка не найдена")
```

# Регулярные выражения в Python

---

- **re.fullmatch**(pattern, string, flags=0) – метод ищет полное соответствие по заданному шаблону. Если совпадения с шаблоном нет, то метод вернет None, иначе Match-объект.

```
s = "Python language"
```

```
print(re.fullmatch("Python language", s))
```

```
<re.Match object; span=(0, 15), match='Python language'>
```

```
print(re.fullmatch("Python", s))
```

```
None
```

# Регулярные выражения в Python

---

- **re.search**(pattern, string, flags=0) – метод ищет в строке первое соответствие заданному шаблону. Если совпадения с шаблоном нет, то метод вернет None, иначе Match-объект.

```
s = "Python language"
```

```
print(re.search("lang", s))
```

```
<re.Match object; span=(7, 11), match='lang'>
```

```
print(re.match("lang", s))
```

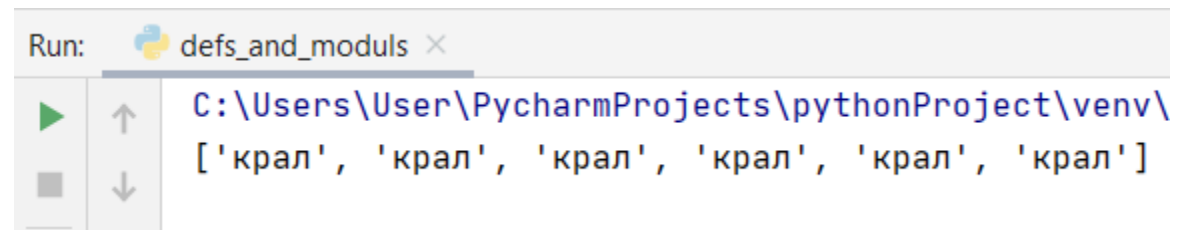
```
None
```

# Регулярные выражения в Python

---

- **re.findall(pattern, string)** – метод возвращает список всех найденных совпадений.

```
import re
s = """Карл у Клары украл кораллы,
    а Клара у Карла украла кларнет.
    Если бы Карл не крал у Клары кораллы,
    то Клара не крала б у Карла кларнет.
    Клара-крякля кралась к Ларе"""
print(re.findall("крал", s))
```



# Регулярные выражения в Python

---

- **re.split(pattern, string, maxsplit=0)** – метод разделяет строку по заданному шаблону.

```
s = "Python language"
print(re.split("n", s))
['Pytho', ' la', 'guage']
```

```
s = "Python language"
print(re.split("n", s, 1))
['Pytho', ' language']
```



# Регулярные выражения в Python

---

- Наиболее часто используемые методы:
  - **re.sub(pattern, repl, string)** – метод ищет шаблон в строке и заменяет его на указанную подстроку. Если шаблон не найден, строка остается неизменной.

```
s = "P3yth4on 21312lan5gu7a9ge123"
```

```
print(re.sub(r"\d", "", s))
```

Python language

```
print(s)
```

P3yth4on 21312lan5gu7a9ge123

# Регулярные выражения в Python

---

- Наиболее часто используемые методы:
  - **re.compile(pattern, flags=0)** – создает объект регулярного выражения в соответствии с заданным шаблоном, который после можно использовать в методах поиска **re.match()**, **re.search()** и др.

# Регулярные выражения в Python

---

- `re.compile(pattern, flags=0)`

```
import re
s = "P3yt4hon 123lan5g7uag9e975"
pattern = re.compile(r"\d")
print(pattern.sub("",s))
print(pattern.split(s))
print(pattern.findall(s))
```

# Регулярные выражения в Python

---

- `re.compile(pattern, flags=0)`

`import re`

`s`

`p`

`p`

`p`

`p`

*Результат:*

Python language

`['P', 'yt', 'hon ', ' ', ' ', 'lan', 'g', 'uag', 'e', ' ', ' ', '']`

`['3', '4', '1', '2', '3', '5', '7', '9', '9', '7', '5']`

# Тема 2.

## Функции, модули, файлы, обработка исключений в Python

# Функции

---

**Создание функции. Инструкция return. Вызов функции. Передача аргументов в функцию. Необязательные аргументы. Функции с переменным числом аргументов. Глобальные и локальные переменные. Анонимные функции. Вложенные функции. Рекурсивные функции.**

# Функции в Python

---

- Python – функциональный язык программирования, поэтому функция в Python является основой при написании программ.
- В Python нет формального разделения подпрограмм на функции и процедуры.
- Фактически, процедура – это функция, возвращающая пустое значение **None**.
-

# Функции в Python

---

- Обычно функции определяется с помощью инструкции **def**.

**Объявление  
процедуры:**

```
def func_name([arguments]):  
    тело процедуры
```

**Объявление  
функции:**

```
def func_name([arguments]):  
    тело функции  
    return <result>
```



# Функции в Python

---

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

```
    f = 1
```

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

```
        f *= i
```

```
    return f
```

```
def fact_rec(n):
```

```
    if n == 1 or n == 0:
```

```
        return 1
```

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

```
def check():
```

```
    n = int(input("input n: "))
```

```
    if fact(n) != fact_rec(n):
```

```
        print("Something is wrong")
```

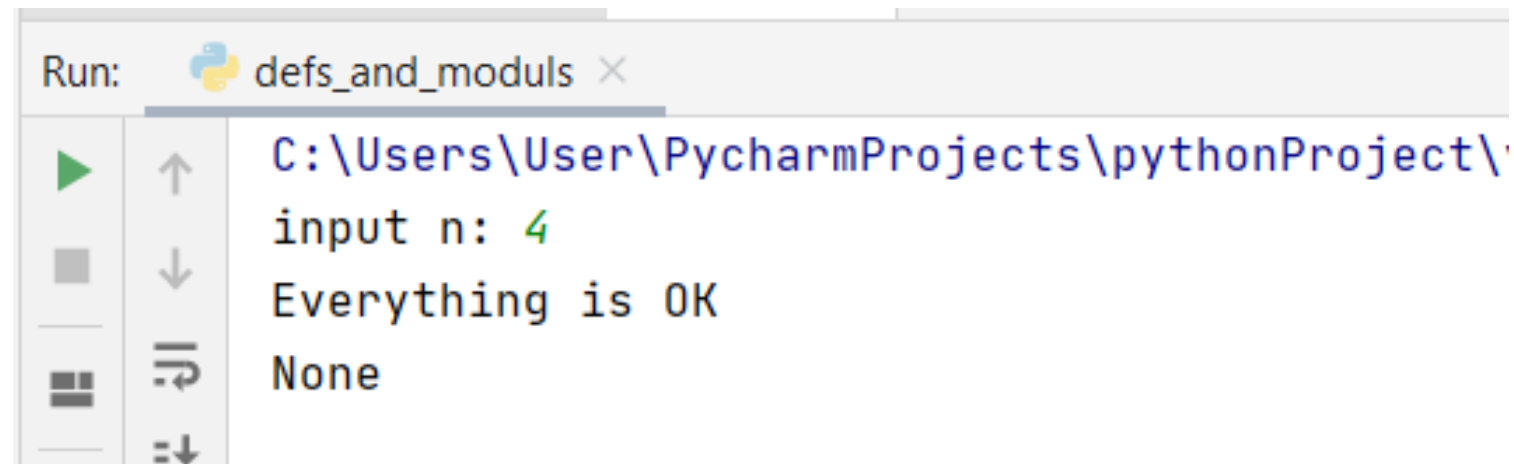
```
    else: print("Everything is OK")
```

# Функции в Python

---

- Если в функции не указан результат выполнения или указан пустой оператор **return**, то возвращается специальное значение **None**.

**print(check())**



The screenshot shows a console window titled "Run: defs\_and\_modules". The output of the program is as follows:

```
C:\Users\User\PycharmProjects\pythonProject\  
input n: 4  
Everything is OK  
None
```

# Функции в Python

---

- **Полиморфизм в Python**
- **Практически каждая операция в Python обладает полиморфизмом: вывод, индексация, операция \* и др.**
- **Если описать функцию, возвращающую результат одной из таких операций, то ее выполнение будет зависеть от типов аргументов, переданных ей.**
- **Т.е. для разных типов аргументов выполняются разные действия для одной и той же операции.**

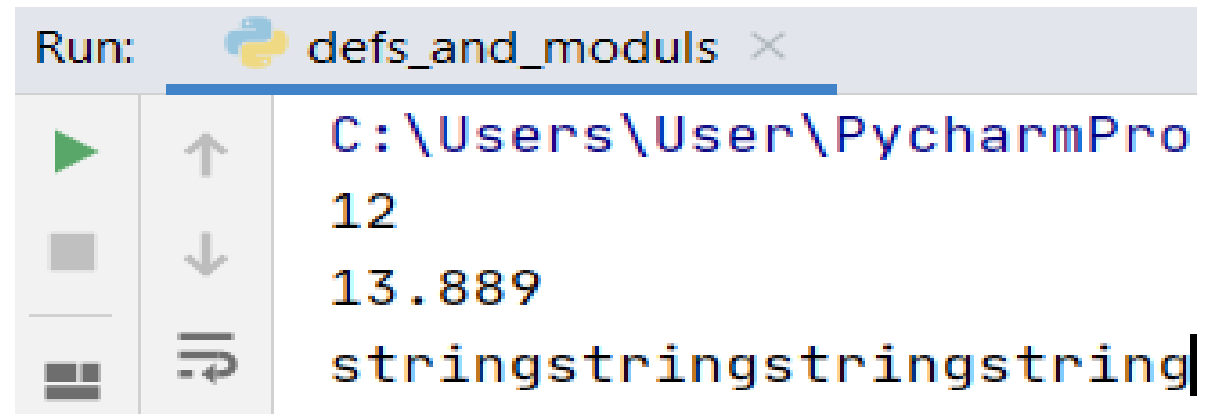
# Функции в Python





---

- **Полиморфизм в Python**

```
def times(a,b):  
    return a*b
```

```
print(times(3, 4))  
print(times(3.23, 4.3))  
print(times("string",4))
```



Run:	defs_and_moduls
	C:\Users\User\PycharmPro
	12
	13.889
	stringstringstringstring

# Функции в Python

---

- **Инструкция def**
- **Важно понимать, что инструкция def – это полноценный исполняемый оператор, который создает объект функции и присваивает его имени.**
- **Следовательно, def может появляться в любом месте кода, где допустимо использование операторов (даже внутри других операторов)**

# Функции в Python

---

- Инструкция `def`
- Пример:
- В зависимости от введенного значения (0 или 1) определить является ли заданное число совершенным (0) или простым (1)

```
quest = int(input("isPerfect - 0 or isPrime - 1? "))  
if quest:
```

*Пример:*

```
def func(x):  
    if x < 2 or x > 2 and x % 2 == 0: return False  
    if x == 2: return True  
    for i in range(3, int(math.sqrt(x) + 1), 2):  
        if (x % i == 0): return False  
    return True
```

```
else:
```

```
def func(x):  
    sum = 1  
    for i in range(2, x // 2 + 1): if x % i == 0: sum += i  
    return sum == x
```

```
x = int(input("x = "))  
print(func(x))
```

# Функции в Python

---

- **Инструкция def**
- В Python допускается также определение вложенных функций.
- Функция, содержащая в себе другую функцию является для нее *объемлющей*.

```
def Fn_1(parameters1):  
    ...  
    def Fn_2(parameters2):  
        ...  
        def Fn_N(parametersN):  
            ...  
            return  
        ...  
        return  
    return
```



# Функции в Python

---

```
def perfect_or_prime(x):  
    def sum_del(x):  
        sum=0  
        for i in range(2,x//2+1):  
            if x%i==0: sum += i  
        return sum  
    if test:  
        return False if sum_del(x)>0 else True  
    else :  
        return False if sum_del(x)+1 != x else True
```

# Функции в Python

---

- Функция в Python – объект, принимающий аргументы и возвращающий значение.
- Объект-функцию можно присвоить другой переменной.

```
def func(x):  
    sum = 1  
    for i in range(2, x // 2+1):    if x % i == 0: sum += i  
    return sum == x  
new_func = func  
print(new_func(x))
```

# Функции в Python

---

- **Область видимости**
- В Python область видимости переменной определяется на основе операции присваивания.
- *Пространство имен* – это место в программном коде, где переменной было присвоено некоторое значение.
- Это пространство имен определяет область видимости данного имени.

# Функции в Python

---

- **Область видимости**
- В Python существуют четыре области видимости:
  - *Встроенная* (включает заведомо определенные имена в модуле встроенных имен `builtins`), такие как: `print`, `range`, `pow` и др.

```
import builtins
built_names = dir(builtins)
for name in built_names:
    print(name)
```

# Функции в Python

---

- **Область видимости**
- В Python существуют четыре области видимости:
  - Глобальная (в пределах модуля)
    - за пределами всех инструкций `def`
    - внутри инструкции `def` с указанием ключевого слова **global**

# Функции в Python

---

- **Область видимости**
- В Python существуют четыре области видимости:
  - Локальная область видимости объемлющих функций
    - переменные, которым присваивается значение в границах объемлющей инструкции `def`
    - переменные, которым присваивается значение внутри вложенной инструкции `def` с указанием ключевого слова **nonlocal**

# Функции в Python

---

- **Область видимости**
- В Python существуют четыре области видимости:
  - **Локальная область видимости функций**
    - переменные, объявленные внутри тела функции (доступ только внутри тела функции)

# Функции в Python

---

- **Область видимости – Правило LEGB**
- **Поиск определения имени в программе на Python осуществляется в следующем порядке:**
  - **Local**
  - **Enclosing**
  - **Global (поиск также происходит и во встроенной области видимости)**
  - **Built-in**



# Функции в Python

---

- Область видимости

```
age = 44 # глобальная переменная age
```

```
def info():
```

```
    x=1 # локальная переменная x
```

```
    print(age)
```

```
info()
```

# Функции в Python

---

- **Область видимости**
- В Python присваивание значения переменной соответствует ее объявлению.
- Это означает, что при присваивании значения переменной внутри функции будет создана одноименная локальная переменная, даже если уже существует одноименная глобальная переменная!

# Функции в Python

---

- Область видимости
- Пример

```
age = 44 # глобальная переменная age
```

```
def info():  
    age = 23 # создана локальная переменная age  
    print(age)
```

```
info()  
print(age)
```

Результат:

23

44

# Функции в Python

---

- **Область видимости**
- Для изменения значения глобальной переменной внутри функции используется команда **global**.

**age=44**

```
def info():  
    global age  
    age = int(input("age = "))  
    print(age)
```

**info()**

# Функции в Python

---

```
def Fn1():
```

```
    x1 = 25 # локальная переменная функции Fn1()
```

```
    def Fn2():
```

```
        x1 = 33 # локальная переменная функции Fn2()
```

```
        def Fn3():
```

```
            nonlocal x1 #обращение к Fn2.x1
```

```
            x1 = 55 # Fn2.x1 = 55
```

```
            Fn3()
```

```
            print('Fn2.x1 = ', x1)
```

```
        Fn2()
```

```
        print('Fn1.x1 = ', x1)
```

```
    Fn1()
```

Результат:

55

25

# Функции в Python

---

```
def Fn1():  
    x1 = 25  
    def Fn2():  
  
        def Fn3():  
            nonlocal x1  
            x1 = 55  
            Fn3()  
            print('Fn2.x1 = ', x1)  
        Fn2()  
        print('Fn1.x1 = ', x1)  
    Fn1()
```

Область видимости переменной x1?

Результат:

55

55

# Функции в Python

---

- Требования к объемлющим функциям, содержащим вложенные функции с **nonlocal**
- Программа должна содержать вложенные функции (одна – объемлющая, другая – вложенная)
- В объемлющей функции должна быть объявлена переменная, которая изменяется во вложенной (эта переменная должна быть объявлена до объявления вложенной функции)

# Функции в Python

---

- **Отличия в правилах поиска `global` и `nonlocal`**
  - **`nonlocal`:**
    - Интерпретатор пропускает локальную область видимости, в которой объявлена инструкция `nonlocal`.
    - Осуществляется поиск в областях видимости объемлющих функций, начиная с ближайшей
  - **`global`:**
    - Интерпретатор осуществляет поиск с глобальной области видимости



# Функции в Python

---

Функция `dir([object_name])`:

- Возвращает имена (переменных, функций), доступных в локальной области видимости, либо атрибуты указанного объекта в алфавитном порядке.
- Результат функции – список имен

# Функции в Python

---

- **Виды аргументов функции:**
  - **Обязательные аргументы**
  - **Аргументы – ключевые слова**
  - **Аргументы по умолчанию**
  - **Аргументы произвольной длины**

# Функции в Python

---

- **Обязательные аргументы:**
  - **Список фактических параметров должен соответствовать (по количеству и по типу) списку формальных параметров (аргументов)**

```
def bigger(a,b):  
    if a > b:  
        print(a)  
    else:  
        print(b)
```

**Вызов:**  
`bigger(5,6)`

# Функции в Python

---

- **Аргументы – ключевые слова:**
  - Используются при вызове функции. Порядок следования фактических параметров может не совпадать с порядком следования формальных параметров

```
def person(name, age):  
    print(name, "is", age, "years old")
```

**Вызовы:**

```
person("John", 23)  
person(age=23, name="John")
```

# Функции в Python

---

- **Аргументы по умолчанию:**

- **Аргумент по умолчанию – это аргумент, для которого задано значение при создании функции.**

```
def space(planet_name, center="Star"):  
    print(planet_name, "is orbiting a", center)
```

**Вызовы:**

```
space("Mars")  
space("Mars", "Black Hole")
```

# Функции в Python

---

- **Аргументы произвольной длины (\*args):**
  - **Используется, если количество аргументов заранее неизвестно**

```
def unknown(*args):  
    for argument in args:  
        print(argument)
```

***Вызовы:***

```
unknown("hello", "world")  
unknown(1, 2, 3, 4, 5)  
unknown()
```

# Функции в Python

---

- **Анонимная функция (лямбда-выражение)**
- В Python лямбда-выражение позволяет создавать анонимные функции – функции, которые не привязаны к имени.
- В анонимной функции:
  - может содержаться только одно выражение
  - могут передаваться сколько угодно аргументов
- Анонимные функции создаются с помощью инструкции `lambda`.

# Функции в Python

---

- Стандартная функция
- Анонимная функция (лямбда-выражение)

```
def sum_arg(a, b):  
    return a + b  
  
print(sum_arg(4, 6))
```

```
sum_arg2 = lambda a, b: a + b  
  
print(sum_arg2(3, 5))
```



# Функции в Python

---

- **Анонимная функция (лямбда-выражение)**
- **Анонимные функции выполняются быстрее, по сравнению с обычными.**
- **Анонимные функции не требуют оператора `return` для возврата результата.**