

Тема 1.

Введение в язык программирования Python. Алгоритмические средства языка Python

продолжение

Условная инструкция if

Логические выражения

- Тип **bool**
- **True**, интерпретируется как 1, **False** – как 0
- Операции сравнения: **>**, **<**, **>=**, **<=**, **==**, **!=**, **<>**
 - к структурам данных применяются рекурсивно
- Логические операторы: **not**, **and**, **or**
- Для Python истинным или ложным может быть не только логическое высказывание, но и объект!

Логические выражения

Пример

>>> 2 > 4

False

>>> 2 > 4 and 45 > 3

False

>>> 2 > 4 or 45 > 3

True

Логические выражения

- **Составные операции сравнения**
- В языке Python есть возможность создавать цепочки из нескольких операций сравнения.
- Цепочка из нескольких операций типа
a op1 b op2 c op3 d ... y opN z
- неявно превращается в форму
a op1 b and b op2 c and c op3 d ... and y opN z

Логические выражения

Проверка принадлежности диапазону:

```
>>> x = 4
```

```
>>> -5 < x < 10
```

```
True
```

```
>>> -5 < x and x < 10
```

```
True
```

Логические выражения

- **Проверка истинности**
 - Любое ненулевое число или непустой объект интерпретируются как **True**
 - Числа, равные 0, пустые объекты и специальный объект **None** – как **False**.
- **Логические операторы `and` и `or` возвращают истинный или ложные объект-операнд.**

Логические выражения

Пример:

>>> ' ' and 2

' '

>>> 0 and 3

0

>>> 5 and 4

4

- **and** вычисляет операнды слева направо и возвращает первый объект, имеющий ложное значение
- Если оба операнда имеют истинное значение, то возвращается крайний правый операнд

Логические выражения

Логические выражения можно комбинировать.

>>> 1 + 3 > 7

False

>>> (1 + 3) > 7

False

>>> 1 + (3 > 7)

1

*Что будет
результатом
выражений?*

1) 4 + (5 and 10)

2) 4 + 5 and 10

Логические выражения

Пример:

>>> ' ' or 2

2

>>> 4 or 6

4

>>> None or 5

5

>>> None or 0

0

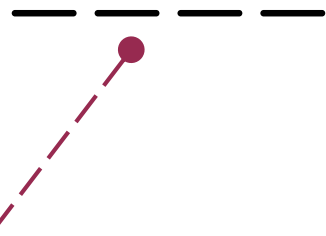
- **or** вычисляет операнды слева направо и возвращает первый объект, имеющий истинное значение
- Если оба операнда имеют ложное значение, то возвращается крайний правый операнд

Логические выражения

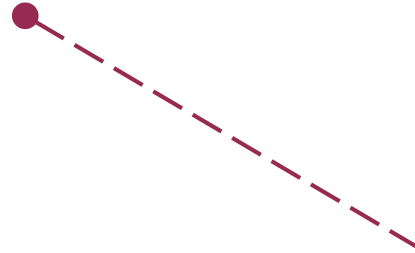
Условная инструкция if

if условие:

<блок выражений>



**4 пробела
или tab**



**1 или несколько
инструкций,
выполняется, если
условие истинно**

Условная инструкция if

```
if условие_1:  
    блок_инструкций_1  
elif условие_2:  
    блок_инструкций_2  
...  
elif условие_N:  
    блок_инструкций_N  
else:  
    блок_инструкций_N+1
```

Полная форма
оператора if

Необязательная часть

Необязательная часть

Условная инструкция if

- Все инструкции одного блока записываются с одинаковым отступом

Пример:

```
if apples > pears:  
    eat_apples(apples - pears)  
    print('We ate some apples')  
else:  
    eat_pears(pears - apples)  
    print('We ate some pears')  
get_more_fruits()
```

Условная инструкция if

- В строке можно разместить несколько инструкций используя точку с запятой в качестве разделителя
- Если блок составной инструкции можно уместить в одну строку его можно вписать в строку заголовка сразу после двоеточия
- Примеры компактной записи

```
a = 1; b = 2; c = a + b; print(c)  
if c < 5: a = 5 ; print(a)
```

Условная инструкция if

- Однако два двоеточия в строке недопустимы
`if c < 5: print('less') else: print('greater') # Ошибка`

Запись в две строки допустима

```
if c < 5: print('less')  
else: print('greater')
```


Условная инструкция if - Пример

```
a,b,c = input("a, b, c: ").split()
a=int(a); b=int(b); c=int(c)
d=b**2-4*a*c
if d>0:
    x1=(-b+d**(0.5))/(2*a)
    x2=(-b-d**(0.5))/(2*a)
    print("x1={:.2f} x2={:.2f}".format(x1,x2))
elif d==0: x = -b/(2*a); print("x=", x)
else: print("Нет решения")
```

*Решение
квадратного
уравнения*

Условная инструкция if

Однострочный оператор if/else

инструкция1 **if** условие **else** инструкция2

- Инструкция1 выполняется если условие истинно, иначе выполняется инструкция2

```
print("A") if a > b else print("B")
```

Условная инструкция if

Вложенные условные конструкции в Python:

- В процессе разработки может возникнуть ситуация, в которой после одной истинной проверки следует сделать еще несколько. В таком случае необходимо использовать вложенные условные конструкции. То есть одну `if...elif...else` конструкцию внутри другой.
- Например:

Условная инструкция if

Пример

```
x, y, op = input("x, y, operation : ").split()
x, y = int(x), int(y)
print(x + y) if op == "+" else \
    print(x - y) if op == "-" else \
    print(x * y) if op == "*" else \
    print(f"{x / y:..3f}") if op == "/" and y != 0 else \
    print("Некорректная операция") if op != "/" else \
    print("Ошибка! Деление на ноль")
```

Метод pass

- **pass** – это пустой оператор.
- Полезен, когда синтаксически выражение требует оператора, но в программе никаких действий выполнять не нужно.

Пример:

```
y = x = int(input("x = "))  
if x > 3:  
    y = x ** 2  
elif x == 3:  
    pass  
else:  
    y = x ** 3  
print(y)
```

Циклы

Циклы

- В Python существуют два вида циклов:
 - Цикл **while**
 - повторение блока кода в зависимости от условия
 - Цикл **for**
 - повторение блока кода для перебираемого набора данных
- Операторы **break** и **continue**
- Часть **else** в циклах – срабатывает в конце цикла, если он был завершен по плану (без **break**)

Цикл **while**

- Цикл **while**
 - тело цикла повторяется пока условие истинно
 - синтаксис:

```
while условие:  
    блок_инструкций  
[else : инструкция]
```


Цикл while

Пример: Определить количество знаков в числе

```
n1 = n = int(input("N = "))
```

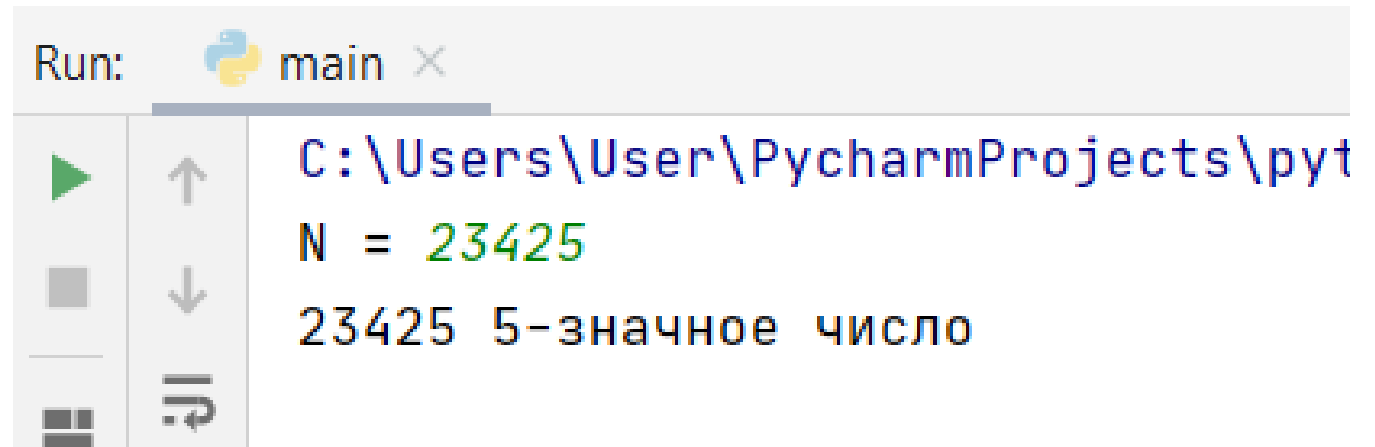
```
k = 0
```

```
while n1 > 0:
```

```
    n1 //= 10
```

```
    k += 1
```

```
else: print(f"{{n}} {{k}}-значное число")
```



```
Run: main x
C:\Users\User\PycharmProjects\pyt
N = 23425
23425 5-значное число
```

Цикл for

- Цикл **for**
 - повторение блока кода для каждого элемента, принадлежащего последовательности (list, tuple, dict, set или str).

```
for item in sequence:  
    statement(s)  
[else : инструкция]
```

Цикл for

- **Примеры использования**

- **Проход по строке (str)**

```
word = input("Your word: ")  
for letter in word: print(letter)
```

- **Проход по списку (list)**

```
seasons = ["winter", "spring", "summer", "autumn"]  
for s in seasons:  
    print(s)
```

Цикл for

Пример: Определить количество гласных букв в введенном слове

```
word = input("Your word: ")
k = 0
for ch in word:
    if ch in "аоеуыюияэ": k += 1
print(f"В {word} {k} гласных букв")
```

Функция `range()`

- Возвращает последовательность чисел
- Как правило, применяется в циклах `for` для указания числа повторений
- *Варианты использования:*
 - `range(stop)` – последовательность чисел от 0 до `stop` с шагом +1
 - `range(start, stop[, step])` – последовательность чисел от `start` до `stop` с шагом `step`

Цикл for

- *Пример:*
- Найти сумму простых чисел, не превышающих 100.

```
import math
k = 0
for x in range(100):
    if x < 2 or x > 2 and x % 2 == 0:
        simple = False
    else:
        for i in range(3, int(math.sqrt(x)), 2):
            if x % i == 0: simple = False; break
        else: simple = True
    if simple: k += x; print(x, " ")
print(f"\nСумма простых элементов = {k}")
```

Цикл for

- *Пример:*
- Найти все двузначные числа, сумма цифр которых кратна 7

```
for x in range(10,99):  
    if (x//10+x%10) % 7 == 0: print(x)
```

Цикл for

- Цикл **for**
 - В начале каждой итерации цикл **for** выполняет присваивания переменным в целевом списке, перезаписывая все предыдущие присвоения которые были сделаны:

```
for i in range(10):  
    print(i, end=" ")  
    i += 2
```

```
C:\Users\User\PycharmProjects\pyth  
0 1 2 3 4 5 6 7 8 9  
Process finished with exit code 0
```


Функции для работы с числами

Функции для работы с числами

- Функции для работы с числами можно разделить на 3 группы:
 - Встроенные функции
 - Функции модуля **math** – не работают с комплексными числами
 - Функции модуля **cmath** – работают с комплексными числами

Встроенные функции для работы с числами

- **abs(x)** – абсолютное значение x (применимо к int, float, complex)
- **bin(x)** , **oct(x)**, **hex(x)** – преобразование целого числа x в двоичную, восьмеричную и шестнадцатеричную формы записи, соответственно
- **divmod(x, y)** – возвращает пару чисел, содержащую результат целочисленного деления и остаток от деления x на y

Встроенные функции для работы с числами

- **max**(x1, x2, ..., xn) – наибольшее из значений x1, x2, ..., xn
- **min**(x1, x2, ..., xn) – наименьшее из значений x1, x2, ..., xn
- **pow**(x, y[, mod]) – возвращает x^y . Если присутствует mod – то результат $x^y \% \text{mod}$
- **round**(number[, ndigits) – возвращает вещественное число, округленное до *ndigits* знаков после запятой

Математические функции – Модуль `math`

- `math.ceil(x)` – возвращает наименьшее целое превосходящее x или равное ему
- `math.comb(n,k)` – число сочетаний k из n
- `math.factorial(x)` – факториал числа x
- `math.floor(x)` - возвращает наибольшее целое не превосходящее x или равное ему
- `math.gcd(x, y)` - возвращает наибольший общий делитель чисел x и y

Математические функции – Модуль math

- **math.trunc(x)** – возвращает целую часть числа (результат – вещественный)
- **math.exp(x)** – e^x
- **math.log(x[, base])** – логарифм числа x по основанию base (по умолчанию – натуральный)
- **math.pow(x, y)** - возвращает x^y в виде вещественного числа
- **math.sqrt(x)** – корень квадратный от x

Математические функции – Модуль math

- **math.acos(x)** – $\arccos(x)$
- **math.asin(x)** – $\arcsin(x)$
- **math.atan(x)** – $\arctg(x)$
- **math.cos(x)** – $\cos(x)$ в радианах
- **math.sin(x)** – $\sin(x)$ в радианах
- **math.tan(x)** – $\tg(x)$ в радианах
- **math.degrees(x)** – x из радианов в градусы
- **math.radians(x)** – x из градусов в радианы

Математические функции – Модуль math

- **math.pi** – число ПИ = 3.141592...
- **math.e** – $\exp(1) = 2.718281...$
- **math.inf** = $+\infty$

Строки

Строки

- **Текстовые данные в Python хранятся в str-объектах.**
- **Строка – неизменяемая последовательность символов Unicode.**
- **Строковые литералы являющиеся частью одного выражения, между которыми имеются только символы разделители, объединяются в единую строку.**
- **Например, ("spam " "eggs") == "spam eggs".**

Строки

- Способы записи строковых литералов:
 - В одинарных кавычках:
'а внутри "можно" поместить обычные'
 - В двойных кавычках:
"а внутри 'можно' поместить одиночные"
 - В тройных кавычках:
 - "'В трёх одиночных кавычках'"
 - """"Three double quotes"""" (для строк документации)

Строки

- **Приведение к строке**
 - Другие типы могут быть приведены к строке при помощи конструктора `str()`: `str(obj)`

Примеры:

`str(10)` \rightarrow `# '10'`

`str(type)` \rightarrow `# "<class 'type'>"`

`str(max)` \rightarrow `# '<built-in function max>'`

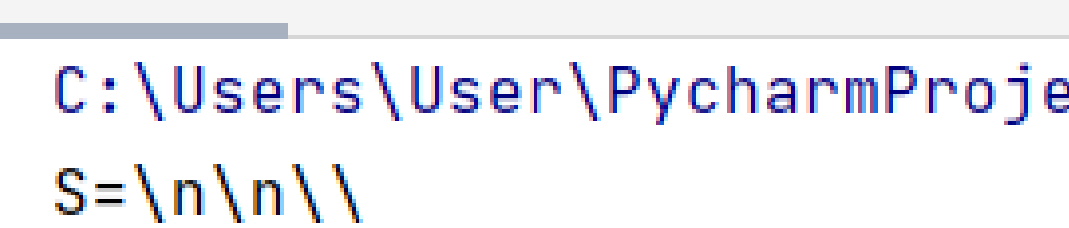
Строки

- **Экранированные последовательности – служебные символы**
 - \n, \t, \r, и др.
- **«Сырые» (raw) строки – подавляют экранирование:**
 - Если перед открывающей кавычкой стоит символ 'r' (в любом регистре), то механизм экранирования подавляется

Строки

- «Сырые» (raw) строки – подавляют экранирование:

```
S = r'\n\n\'
print(S)
S1 = '\n\n\'
print(S1)
```



```
main x
C:\Users\User\PycharmProjects
S=\n\n\\
S1=
\
```

Строки

- «Сырые» (raw) строки :
- В raw-строках также можно добавлять кавычки. Однако символ \ будет также включен в результат.
- Пример правильной строки: `r"\\" → \`
 - Количество \ в конце raw-строки должно быть четным (т.к. каждый \ требует следующего за ним символа экранирования)
- Примеры некорректных строк, `r"\`, `r"\n\`, ...

Строки. Функции и методы строк

- **Базовые операции :**

- *Конкатенация*
(сложение) – операция

+

```
s1 = 'spam'  
s2 = "eggs"  
print(s1+s2)
```

Результат: 'spameggs '

- *Дублирование строки* *

```
s1 = 'spam'  
print(s1*3)
```

Результат: 'spamspamspam '

Строки. Функции и методы строк

- **Базовые операции :**

- *Длина строки – функция **len***

```
s1 = 'spam'  
print(len(s1))  
Результат: 4
```

- *Доступ по индексу:
индекс может быть как
положительный, так и
отрицательный*

```
s1 = 'spam'  
print(s1[1], " ", s1[-1])  
Результат: 'p' 'm'
```

Строки. Функции и методы строк

- **Базовые операции :**
 - *Извлечение среза* – оператор **[X:Y]**,
X – начало среза (=0, по умолчанию),
Y – окончание (не включается в результат) – (=длине строки, по умолчанию)

```
s = 'python version'
```

```
print(s[3:5])
```

```
print(s[2:-2])
```

```
print(s[:6])
```

```
print(s[1:])
```

```
print(s[:])
```

ho

thon versi

python

ython version

python version

Строки. Функции и методы строк

- **Базовые операции :**

- *Извлечение среза* – оператор **[X:Y:Z]**,
X – начало среза,
Y – окончание
Z – шаг

```
s = 'python version'
```

```
print(s[::-1])
```

```
print(s[2:5:-2])
```

```
print(s[9:2:-2])
```

```
print(s[2::3])
```

```
'noiserv nohtyp'
```

```
“
```

```
'rvnh'
```

```
'tnei'
```

Строки. Функции и методы строк

- **Базовые операции :**

- **ord(символ)** – возвращает ASCII код символа
- **chr(число)** – возвращает символ по ASCII коду

```
print(ord("a"))  
print(chr(68))
```

97

D

Строки. Примеры

Пример_1

- Определить, является ли строка палиндромом.

```
s = input("Введите строку: ")
palindrom = True
for i in range(len(s) // 2+1):
    if s[i] != s[len(s) - i - 1]:
        palindrom = False
        break
print("Да") if palindrom else print("Нет")
```

Строки. Функции и методы строк

Методы строк

- **Вследствие того, что строки в Python неизменяемые, то все методы могут лишь создавать новую строку.**
- **Для изменения значения строковой переменной, требуется присваивать ей значение, возвращаемое строковыми методами.**

Строки. Функции и методы строк

Методы строк

- **S.find(str, [start],[end])** – поиск подстроки str в строке S. Возвращает номер первого вхождения или -1
- **S.rfind(str, [start],[end])** – поиск подстроки str в строке S. Возвращает номер последнего вхождения или -1

Строки. Функции и методы строк

Методы строк

- **S.index(str, [start],[end])** – поиск подстроки str в строке S. Возвращает номер первого вхождения или вызывает **ValueError**
- **S.rindex(str, [start],[end])** – поиск подстроки str в строке S. Возвращает номер последнего вхождения или вызывает **ValueError**

Строки. Функции и методы строк

Методы строк

- **S.replace(str1, str2 [,n])** – в строке подстрока str1 заменяется на str2 (n замен)
- **S.split(символ)** – Разбиение строки по разделителю
- **S.isdigit()** – Состоит ли строка только из цифр
- **S.isalpha()** – Состоит ли строка только из букв
- **S.isalnum()** – Состоит ли строка только из цифр или букв

Строки. Примеры

Пример_2

- Дана строка
символов.
Определить
количество цифр и
гласных букв в ней

```
s = input("Введите строку: ")
d=g=0
for c in s:
    if c.isdigit():
        d+=1
    elif c in "аоиеуі" :
        g+=1;
print(f"Гласных {g}, Цифр {d}")
```

Строки. Функции и методы строк

Методы строк

- **S.islower(), S.isupper()** – Состоит ли строка только из символов нижнего (верхнего) регистра
- **S.isspace()** – Состоит ли строка только из символов-разделителей
- **S.istitle()** – Начинаются ли все слова в строке с заглавной буквы

Строки. Функции и методы строк

Методы строк

- **S.lower()**, **S.upper()** – преобразование строки нижнему (верхнему) регистру
- **S.startswith(str)**, **S.endswith(str)** – Начинается (заканчивается) ли строка подстрокой str
- **S.join(список)** – сборка строки из списка с разделителем S
- **S.capitalize()** – переводит первый символ строки в верхний регистр, а все остальные в нижний

Строки. Примеры

Пример_3

- Преобразовать символы строки к верхнему или нижнему регистру в зависимости от того, символов в каком регистре в строке больше.

```
s = input("Введите строку: ")
u = l = 0
for i in range(len(s)):
    if s[i].isupper():
        u += 1
    else:
        l += 1
s = s.upper() if u > l else s.lower()
print(s)
```

Строки. Функции и методы строк

Методы строк

- **S.center(width, [fill])**, возвращает отцентрированную строку, по краям которой стоит символ fill (пробел по умолчанию)
- **S.count(str, [start], [end])** – количество непересекающихся вхождений подстроки в диапазоне [start, end] (0 и длина строки по умолчанию)

Строки. Примеры

Пример_4

- Дан текст из слов, разделенных пробелом. Определить, можно ли из букв самого длинного слова составить заданное слово

```
s = input("Текст: ").split(" ")
maxword=""
for word in s:
    if len(word) > len(maxword):
        maxword = word
w = input("Введите слово: ")
f=True
for c in w:
    if w.count(c)>maxword.count(c):
        f=False
        break
print("Да") if f else print("Нет")
```

Строки. Функции и методы строк

Методы строк

- **`S.expandtabs([tabsize])`** – возвращает копию строки, в которой все символы табуляции заменяются одним или несколькими пробелами, в зависимости от текущего столбца. Если `TabSize` не указан, размер табуляции полагается равным 8 пробелам.

Строки. Функции и методы строк

Методы строк

- **S.lstrip([chars])** – удаление символов (chars) в начале строки (пробелы по умолчанию)
- **S.rstrip([chars])** – удаление символов (chars) в конце строки (пробелы по умолчанию)
- **S.strip([chars])** – удаление символов (chars) в начале и в конце строки (пробелы по умолчанию)

Строки. Функции и методы строк

Методы строк

- **S.partition(шаблон)** – возвращает кортеж, содержащий часть перед первым шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий саму строку, а затем две пустые строки
- **S.rpartition(шаблон)** – возвращает кортеж, содержащий часть перед последним шаблоном, сам шаблон, и часть перед шаблона. Если шаблон не найден, возвращается кортеж, содержащий две пустые строки, а затем саму строку

Строки. Функции и методы строк

Методы строк

- **S.swapcase()** – переводит символы из нижнего регистра в верхний, а из верхнего регистра – в нижний
- **S.title()** – первую букву каждого слова переводит в верхний регистр, а все остальные в нижний
- **S.zfill(width)** – делает длину строки не меньшей width, по необходимости заполняя первые символы нулями ("0")

Строки. Функции и методы строк

Методы строк

- **S.ljust**(width, fillchar=" ") – делает длину строки не меньшей width, по необходимости заполняя последние символы символом fillchar
- **S.rjust**(width, fillchar=" ") – делает длину строки не меньшей width, по необходимости заполняя первые символы символом fillchar
- **S.format**(*args, **kwargs)

Строки. Функции и методы строк

Пример

- Дан текст из слов, разделенных пробелами. Найти самое короткое слово текста, встречающееся в тексте более 2 раз. Вывести на экран первое и последнее вхождения данного слова.
- Удалить все вхождения этого слова в текст.
- Все оставшиеся слова написать с заглавной буквы.

Строки. Функции и методы строк

Пример

```
st = input("Input string: ")
words=st.split(); minword=st
for word in words:
    c=0
    for i in range(len(words)):
        if word==words[i]: c+=1
    if c>2 and len(word)<len(minword):
        minword=word
```

Строки. Функции и методы строк

Пример (продолжение)

```
import re
print(minword)
print(f"{st.find(minword)} {st.rfind(minword)}")

pattern = re.compile(r"\b"+minword+r"\b")
s1=" ".join(pattern.split(st))
print(s1.title())
```