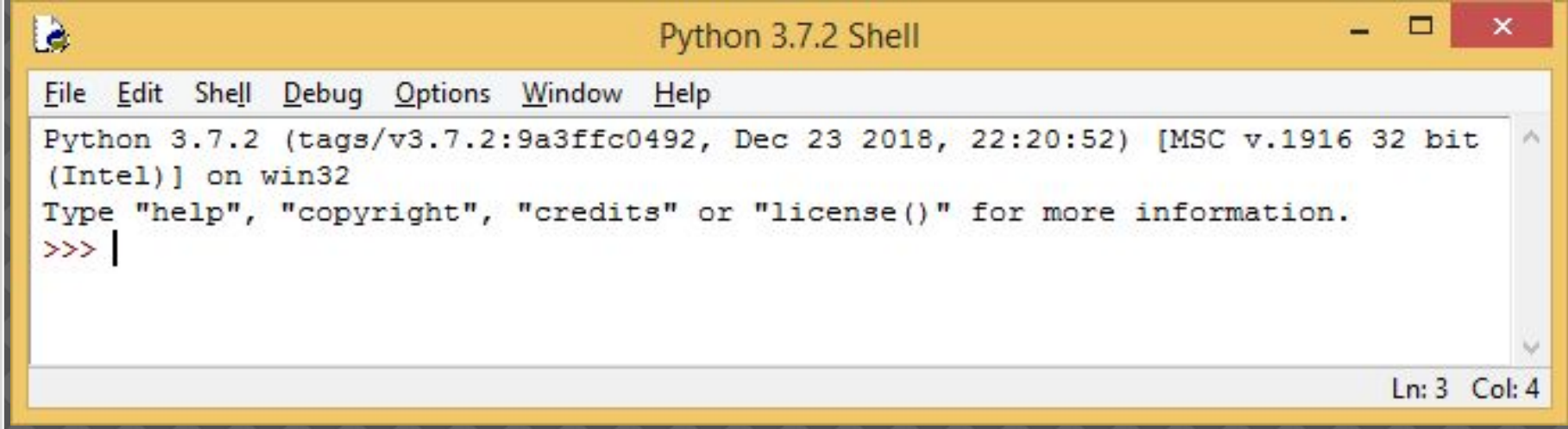


PYTHON

Логический тип данных и операции Управляющие конструкции

РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПОСЛЕДНЕЙ ИНСТРУКЦИИ

- Результат выполнения последней инструкции сохраняется в переменной _ (одно подчеркивание), что позволяет производить дальнейшие вычисления

```
>>> 125*3
375
>>> _+36
411
```

Обозначение сравнения

- На естественном языке (например, русском) мы обозначаем сравнение словами "равно", "больше", "меньше" и им подобными

ЛОГИЧЕСКИЕ ОПЕРАТОРЫ

- В языках программирования используются специальные знаки, подобные тем, которые используются в математике:
  - > (больше),
  - < (меньше),
  - >= (больше или равно),
  - <= (меньше или равно),
  - == (равно),
  - != (не равно)

ЛОГИЧЕСКИЙ ТИП ДАННЫХ

- Логический тип данных может принимать значения "истина" (True) или "ложь" (False).
- Простое логическое выражение имеет вид <арифметическое выражение> <знак сравнения> <арифметическое выражение>

Сравнения в логических выражениях

Знак сравнения	Описание
==	Равно
!=	Не равно
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно

Допустимы двойные сравнения, например x < y <= z

У всех операций сравнения одинаковый приоритет, меньший, чем у арифметических операций (т.е. сначала выполняются все арифметические операции и только потом сравнения)

УСЛОВНЫЕ ОПЕРАТОРЫ

- Условные операторы позволяют в зависимости от значения логического выражения выполнить отдельный участок программы или, наоборот, не выполнить его.
- Логические выражения возвращают только два значения: True (истина) или False (ложь), которые ведут себя как целые числа 1 и 0 соответственно.

```
>>> True + 2 # Эквивалентно 1 + 2
3
>>> False + 2 # Эквивалентно 0 + 2
2
```

X < Y < Z

- В Питоне допустимы и логические выражения, содержащие несколько знаков сравнения, например x < y < z. При этом все сравнения обладают одинаковым приоритетом, который меньше, чем у любой арифметической операции.
- Результат вычисления логического выражения можно сохранять в переменную, которая будет иметь тип **bool**. Переменные такого типа, как и числа и строки, являются неизменяемыми объектами.
- Строки также могут сравниваться между собой. При этом сравнение происходит в лексикографическом порядке (как упорядочены слова в словаре).

ОПЕРАТОРЫ ТОЖДЕСТВЕННОСТИ В PYTHON

Оператор	Значение	Пример
is	Возвращает true если переменные являются одним объектом (имеют одинаковый адрес памяти)	x is y
is not	Возвращает true если переменные разные	x is not y

ОПЕРАТОРЫ ТОЖДЕСТВЕННОСТИ

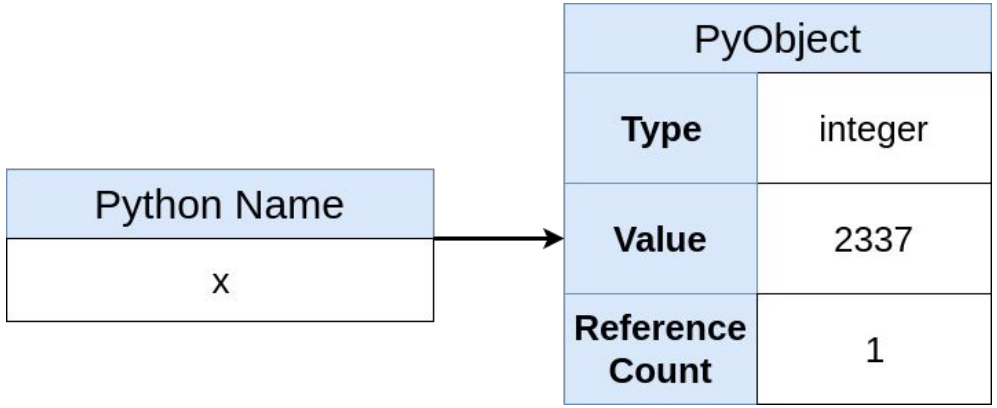
- is - проверяет, ссылаются ли две переменные на один и тот же объект. Если переменные ссылаются на один и тот же объект, то оператор is возвращает значение True:

```
>>> x = y = [1, 2]
>>> x is y
True
>>> x = [1, 2]
>>> y = [1, 2]
>>> x is y
False
```

- is not - проверяет, ссылаются ли две переменные на разные объекты. Если это так, возвращается значение True:

```
>>> x = y = [1, 2]
>>> x is not y
False
>>> x = [1, 2]
>>> y = [1, 2]
>>> x is not y
True
```

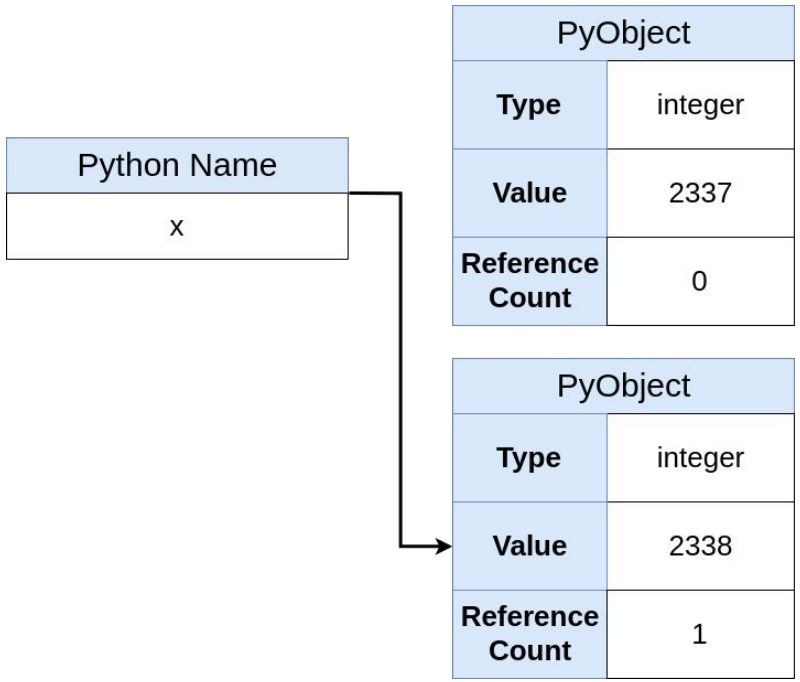
X = 2337



Python-имя x не владеет напрямую каким-либо адресом в памяти

1. Создаётся PyObject.
2. Числу для PyObject’а присваивается typecode.
3. 2337 присваивается значение для PyObject’а.
4. Создаётся имя x.
5. x указывает на новый PyObject.
6. Счётчик ссылок PyObject’а увеличивается на 1.

X = 2337

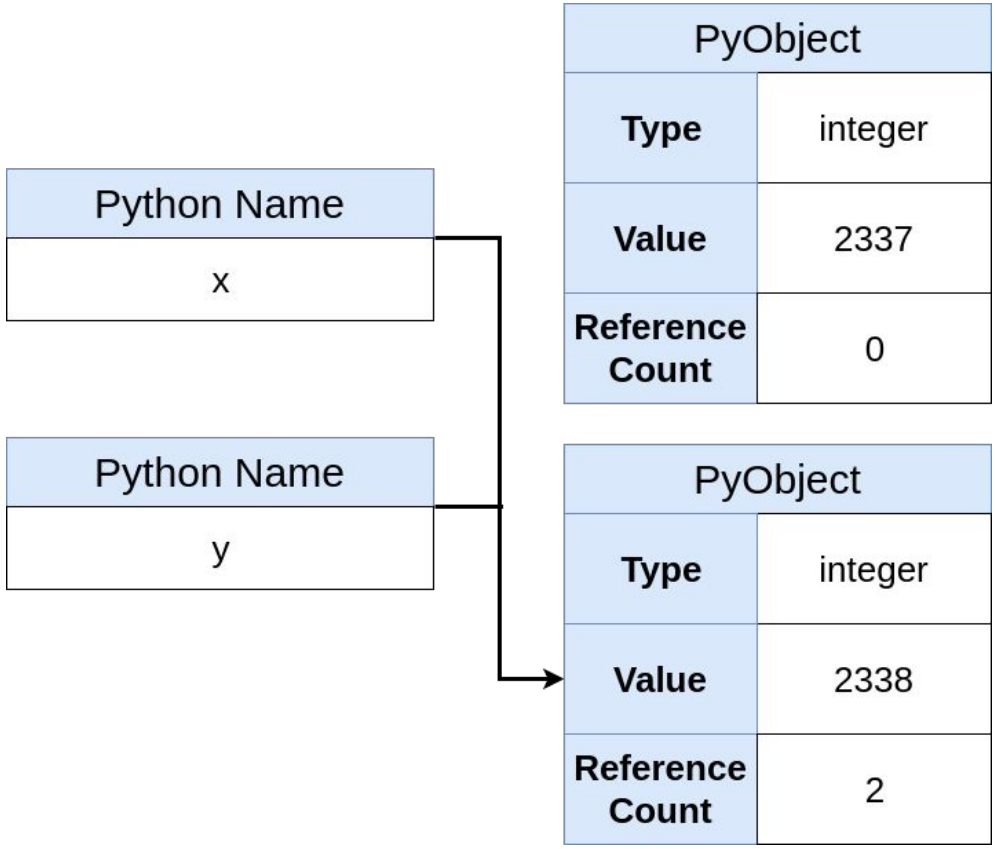


1. Создаётся новый PyObject.
2. Числу для PyObject’а присваивается typecode.
3. 2337 присваивается значение для PyObject’а.
4. Создаётся имя x
5. x указывает на новый PyObject.
6. Счётчик ссылок нового PyObject увеличивается на 1.

команда x = 2338 является не присваиванием, а, скорее, привязкой (binding) имени x к ссылке

Y = X

Y IS X TRUE



- В памяти появится новое имя, но не обязательно новый объект
- Новый Python-объект **не** создан, создано только новое имя, которое указывает на тот же объект. Кроме того, счётчик ссылок объекта увеличился на 1

ОПЕРАТОРЫ ПРИНАДЛЕЖНОСТИ В PYTHON

Оператор	Значение	Пример
in	Считается истиной (true), если находит переменную в заданной последовательности, и ложью (false) в противном случае	x in y
not in	Возвращает True если последовательность не присутствует в объекте	x not in y

ПРИМЕР

```
a = int(input())
list = [1, 2, 3, 4, 5 ];
if ( a in list ):
    print ("Число есть")
else:
    print("Числа нет")
```

```
3
Число есть
>>>
============================ RESTART: D:/python/pr3.py ==

6
Числа нет
>>>
```

NUMBER % 2 == 0

- Одним из примеров использования логического выражения является проверка на делимость. Например, чтобы проверить, является ли число четным, необходимо сравнить остаток от деления этого числа на два с нулём:

```
File  Edit  Format  Run  Op
a = int(input())
isChet = a % 2 == 0
print (isChet)
```

```
>>>
= RESTART:
36
True
>>>
= RESTART:
11
False
```

Определите, является ли выражение логическим

- x = 12 - 5
- x == 4
- x == 7
- x != 7
- x != 4
- x > 5
- x < 5
- x <= 6
- x >= 6

Сложные логические выражения

- Часто требуется получить логический тип ответа ("Да" или "Нет") в зависимости от результата выполнения двух простых выражений, например, переменная больше 12 и меньше 35.

ЛОГИЧЕСКИЕ ОПЕРАТОРЫ В PYTHON

Оператор	Значение	Пример
and	Возвращает значение True если оба утверждения верны	x < 5 and x < 10
or	Возвращает True если одно из утверждений верно	x < 5 or x < 4
not	Меняет результат, возвращает False если результат True	not(x < 5 and x < 10)

Связки "И", "ИЛИ" и "НЕ"

Логические операции

Для записи логических выражений часто требуются логические операции «И», «ИЛИ» и «НЕ»

Обозначаются как «and», «or» и «not» соответственно

«And» и «or» – бинарные операции (записываются между выражениями)

«not» – унарная (записывается перед выражением)

У всех логических операций приоритет ниже, чем у сравнений. Среди логических операций самый высокий приоритет у «not», затем у «and» и самый низкий у «or»

- Все логические операции имеют приоритет ниже, чем операции сравнения (а значит, и ниже чем арифметические операции).
- Среди логических операций наивысший приоритет имеет операция not, затем идет and и наименьший приоритет имеет операция or.
- На порядок выполнения операций можно влиять с помощью скобок, как и в арифметических выражениях.

ОПЕРАТОРЫ СРАВНЕНИЯ

- Операторы сравнения в порядке убывания приоритета:

1. <, >, <=, >=, ==, !=, is, is not, in, not in.
2. not (логическое отрицание).
3. and (логическое И).
4. or (логическое ИЛИ).

Являются ли приведенный примеры сложными логическими выражениями

- x = 8
- y = 13
- x == 8 **and** y < 15
- x > 8 **and** y < 15
- x != 0 **or** y >15

ЗАДАНИЕ

1. Присвойте двум переменных числовые значения. Составьте с помощью этих переменных два сложных выражения с применением оператора and, которые должны выдавать разные результаты (true и false)
2. Составьте с помощью этих переменных четыре сложных выражения с применением оператора or, которые должны выдавать разные результаты (true и false)

ОПЕРАТОР ВЕТВЛЕНИЯ IF...ELSE

- Оператор ветвления if...else позволяет в зависимости от значения логического выражения выполнить отдельный участок программы или, наоборот, не выполнить его. Оператор имеет следующий формат:

```
if <Логическое выражение>:
    <Блок, выполняемый, если условие истинно>
[elif <Логическое выражение>:
    <Блок, выполняемый, если условие истинно>
]
[else:
    <Блок, выполняемый, если все условия ложны>
]
```

ПРИМЕР

```
# -*- coding: utf-8 -*-
x = int(input("Введите число: "))
if x % 2 == 0:
    print(x, " - четное число")
else:
    print(x, " - нечетное число")
input ()
```

УСЛОВНЫЙ ОПЕРАТОР

```
x = int(input())
if x < 0:
    x = -x
print(x)
```

- Условный оператор позволяет выполнять действия в зависимости от того, выполнено условие или нет.
- Записывается условный оператор как
  - "if <логическое выражение>:‘’,
  - далее следует блок команд, который будет выполнен только если логическое выражение приняло значение True.
- Блок команд, который будет выполняться, выделяется отступами в 4 пробела (в IDE можно нажимать клавишу tab).

if *a* *логический_оператор* *b* :

выражения, выполняемые при результате True в логическом выражении

УСЛОВНЫЙ ОПЕРАТОР

- При выполнении программного кода некоторые его участки могут быть пропущены
- Если часть кода должна выполняться лишь при определенном значении конкретной переменной, то используется следующая конструкция:

if *a* *логический_оператор* *b* :

выражения, выполняемые при результате True в логическом выражении

- Логическое значение можно хранить в переменной или сравнивать со значением **True** и **False**.
- Значение True возвращают объекты – число, **не равное 0**, например, bool(12), непустой объект, например, bool(“0”).
- Значение False возвращают объекты bool(‘’), bool([]), bool(()), None.

УСЛОВНЫЕ ОПЕРАТОРЫ

- Логическое значение можно сохранить в переменной:

```
>>> x = True; y = False
>>> x, y
(True, False)
```

- Значение True возвращает следующие объекты:

- любое число, не равное нулю:

```
>>> bool(1), bool(20), bool(-20)
(True, True, True)
>>> bool(1.0), bool(0.1), bool(-20.0)
(True, True, True)
```

- не пустой объект:

```
>>> bool("0"), bool([0, None]), bool((None, )), bool({'x': 5})
(True, True, True, True)
```

УСЛОВНЫЕ ОПЕРАТОРЫ

- Следующие объекты интерпретируются как False:
  - число, равное нулю:

```
>>> bool(0), bool(0.0)
(False, False)
```

  - пустой объект:

```
>>> bool(""), bool([]), bool(())
(False, False, False)
```

  - значение None:

```
>>> bool(None)
False
```

ОПЕРАТОР ELSE: (ИНАЧЕ)

- Все команды, которые выполняются в блоке else, должны быть также записаны с отступом.
- Else должен следовать сразу за блоком команд if, без промежуточных команд, выполняемых безусловно.
- Else без соответствующего if'а не имеет смысла.

```
x = int(input())
if x >= 0:
    print(x)
else:
    print(-x)
```

ВЕТВЛЕНИЕ. УСЛОВНЫЙ ОПЕРАТОР

```
>>> a = 10
>>> if a > 0:
...     a = a * 0.7
...
>>> a
7.0
```

Здесь был нажат Tab

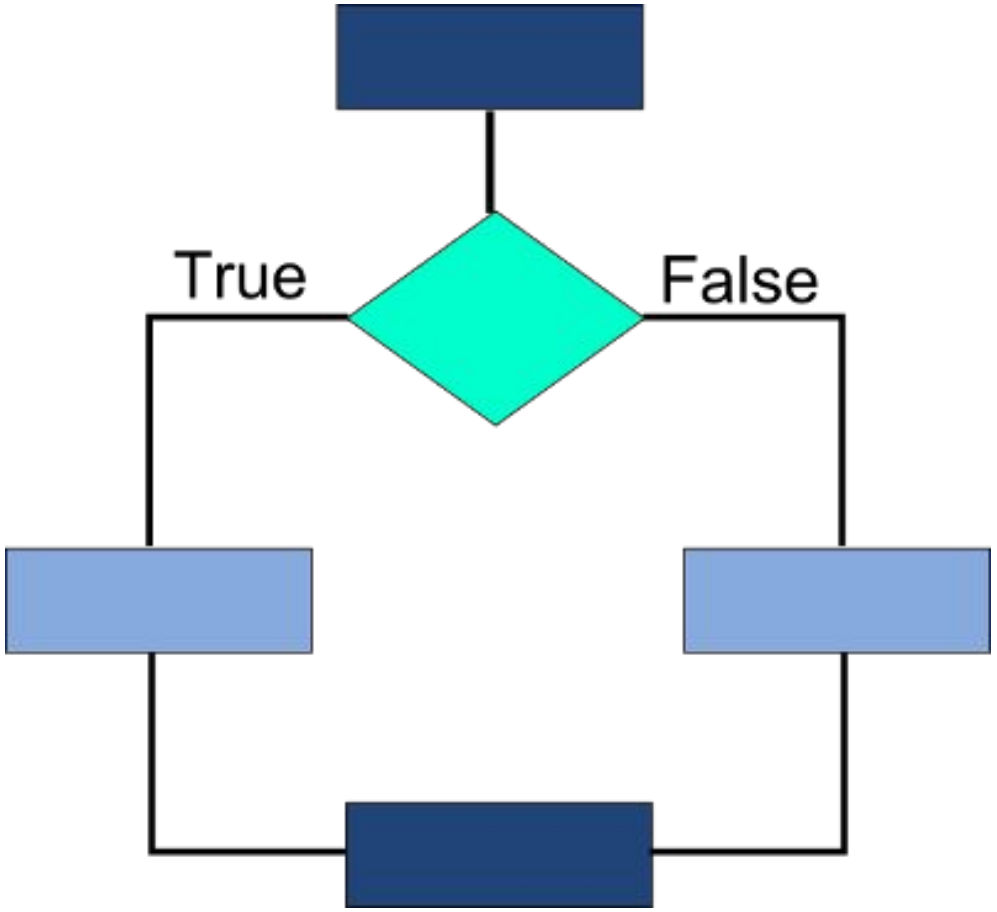
Здесь был нажат Enter

- Структуру программы можно изобразить следующим образом:

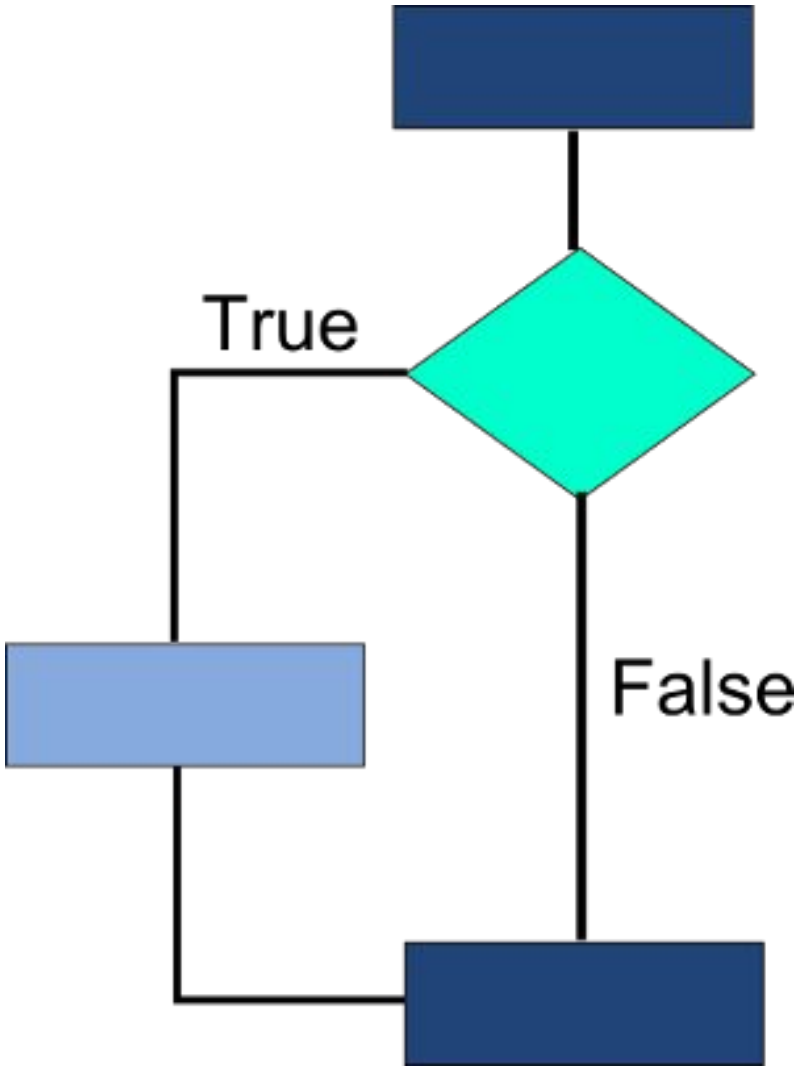
ПРИВЕСТИ ПРИМЕР

```
>>> x
5
>>> if x%2==0:
        print(x," четное число")
else:
        print(x," нечетное число")


5  нечетное число
```



ПРИВЕДИТЕ ПРИМЕР



ОПЕРАТОР ВЕТВЛЕНИЯ IF..ELIF..ELSE

```
>>> if os!="":
        if os=="1":
                print('Windows')
        elif os=="2":
                print('Linux')
        elif os=="3":
                print('MacOS')
        else:
                print('Другая')
else:
        print('Вы не ввели число')
```

ЗАДАНИЕ

- Присвойте двум переменным a и b два числовых значения
- Выведите на экран значение True или False в зависимости от того, больше ли первое число второго или нет

- Если после if записано не логическое выражение, то оно будет приведено к логическому, как если бы от него была вызвана функция **bool**
- *Однако, злоупотреблять этим не следует, т.к. это ухудшает читаемость кода*

ВЛОЖЕННЫЙ УСЛОВНЫЙ ОПЕРАТОР

- Внутри блока команд могут находиться другие условные операторы.
- Если вложенных условных операторов несколько, то, к какому из них относится else, можно понять по отступу. Отступ у else должен быть такой же, как у if, к которому он относится.

```
eyes = int(input())
legs = int(input())
if eyes >= 8:
    if legs == 8:
        print("spider")
    else:
        print("scallop")
else:
    if legs == 6:
        print("bug")
    else:
        print("cat")
```

КОНСТРУКЦИЯ "ИНАЧЕ-ЕСЛИ" ELIF

- В некоторых ситуациях необходимо осуществить выбор больше чем из двух вариантов, которые могут быть обработаны с помощью if-else.

Конструкций elif может быть несколько, условия проверяются последовательно. Как только условие выполнено, запускается соответствующий этому условию блок команд и дальнейшая проверка не выполняется. Блок else

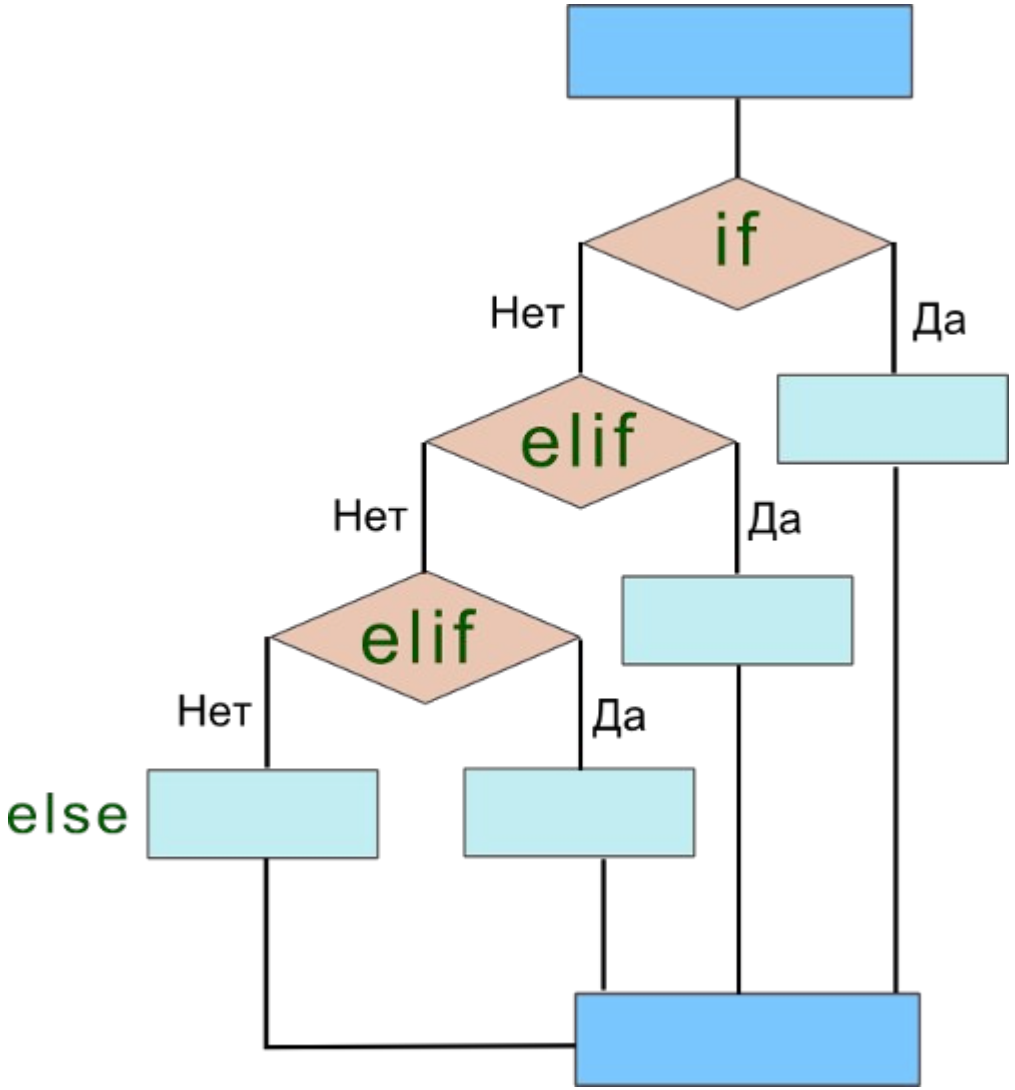
```
number = int(input()
if number == 1:
    print('One')
elif number == 2:
    print('Two')
else:
    print('Other')
```

МНОЖЕСТВЕННОЕ ВЕТВЛЕНИЕ: IF-ELIF-ELSE

```
old = int(input('Ваш возраст: '))

print('Рекомендовано:', end=' ')

if 3 <= old < 6:
    print('''Заяц в лабиринте''')
elif 6 <= old < 12:
    print('''Марсианин''')
elif 12 <= old < 16:
    print('''Загадочный остров''')
elif 16 <= old:
    print('''Поток сознания''')
```



ЗАДАНИЕ

- Напишите программный код, в котором в случае, если значение некой переменной больше 0, выводилось бы специальное сообщение.
- С помощью ветки else добавьте код таким образом, чтобы в зависимости от значения переменной, выводилась либо 1, либо -1.

ЦИКЛ FOR

- Цикл for в языке программирования Python предназначен для перебора элементов структур данных и некоторых других объектов.
- В цикле **for** указывается переменная и множество значений, по которому будет пробегать переменная.
- Множество значений может быть задано списком, кортежем, строкой или диапазоном.

```
>>> spisok = [10, 40, 20, 30]
>>> for element in spisok:
        print(element + 2)

12
42
22
32
>>> spisok
[10, 40, 20, 30]
```

ФУНКЦИЯ RANGE()

- Для повторения цикла некоторое заданное число раз n можно использовать цикл **for** вместе с функцией **range**:

```
for i in range(n):
    тело цикла
```

- В качестве n может использоваться числовая константа, переменная или произвольное арифметическое выражение (например, 2 ** 10). Если значение n равно нулю или отрицательное, то тело цикла не выполнится ни разу

Функция range(a,b)

- Если задать цикл таким образом:

```
for i in range(a, b):
    тело цикла
```

- То индексная переменная i будет принимать значения от a до b – 1, то есть первый параметр функции range, вызываемой с двумя параметрами:
  - Первый параметр задает начальное значение индексной переменной
  - Второй параметр — значение, которая индексная переменная принимать не будет
- Если a≥b, то цикл не будет выполнен ни разу

ПРИМЕР

```
sum = 0
for i in range(1, n + 1):
    sum += i
```

ФУНКЦИЯ RANGE(A,B,H)

- Чтобы организовать цикл, в котором индексная переменная будет уменьшаться, необходимо использовать функцию range с тремя параметрами.
  - Первый параметр задает начальное значение индексной переменной
  - Второй параметр — значение, до которого будет изменяться индексная переменная (не включая его!)
  - Третий параметр — величину изменения индексной переменной
- Цикл по всем нечетным числам от 1 до 99 можно при помощи функции **range(1, 100, 2)**
- Цикл по всем числам от 100 до 1 можно при помощи **range(100, 0,-1)**.

FOR I IN RANGE(A, B, D)

- Цикл for i in range(a, b, d)
  - при d > 0 задает значения индексной переменной i = a, i = a + d, i = a + 2 * d и так для всех значений, для которых i < b.
  - при d < 0 переменная цикла принимает все значения i > b.

ФУНКЦИЯ RANGE()

- Range" переводится как "диапазон". Она может принимать один, два или три аргумента.
- Если задан только один, то генерируются числа от 0 до указанного числа, не включая его.
- Если заданы два, то числа генерируются от первого до второго, не включая его. Если заданы три, то третье число – это шаг.

```
>>> a=range(-10,10)
>>> a
range(-10, 10)
>>> type(a)
<class 'range'>
```

```
>>> a[0]
-10
>>> a[-1]
9
>>> a[9]
-1
```

```
>>> a[0]=10
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    a[0]=10
TypeError: 'range' object does not support item assignment
```

ЗАДАНИЕ

- Дано натуральное число. Требуется определить, является ли год с данным номером високосным. Если год является високосным, то выведите YES, иначе выведите NO.
  - В соответствии с григорианским календарем, год является високосным, если его номер кратен 4, но не кратен 100, а также если он кратен 400

Циклы

- Циклы — это инструкции, выполняющие одну и ту же последовательность действий многократно
- Ходьба человека — вполне циклическое явление: шаг левой, шаг правой, снова левой-правой и т. д., пока не будет достигнута определенная цель

ЦИКЛЫ

- Цикл for используется, когда известно количество повторений цикла или известен набор значений, для которых должно повторяться действие
- Однако данная информация не всегда известна

ЦИКЛ WHILE

```
i = 1    #  <Начальное значение>
while  i  <  11: #  <Условие>
    print(i) #  <Инструкции>
    i += 1    # <Приращение>
```

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===================== RESTART: F:\Python\Лекция 1\pr28_1.py ====================
1
2
3
4
5
6
7
8
9
10
>>>
```

WHILE

while a логический_оператор b:

действие(я)
изменение a

```
>>> arr
[1, 1, 2, 3, 4, 5, 6]
>>> i,count=0,len(arr)
>>> while i<count:
        arr[i]*=2
        i+=1
```

```
>>> arr
[2, 2, 4, 6, 8, 10, 12]
```

ЦИКЛ WHILE - ЦИКЛ С УСЛОВИЕМ

- Выполнение инструкций в цикле while продолжается до тех пор, пока логическое выражение истинно. Цикл while имеет следующий формат:

```
<Начальное значение>
while <Условие>:
    <Инструкции>
    <Приращение>
[else:
    <Блок, выполняемый, если не использовался оператор break>
]
```

ПРИМЕР

- В багажник автомобиля грузят овощи и фрукты с дачи: картофель, капусту, морковь, яблоки, груши и др. Объем багажника равен 350л. Продукты кладут последовательно, объём каждого груза известен в литрах. Нужно сказать в какой момент (назвать номер груза) багажник переполнится

```
s = 0
n = 0
while s < 350:
    x = int ( input ())
    s = s + x
    n = n + 1
print ( n)
```

```
"C:\Arch2121\НИЯУ МИФИ\Task 1-1\djang
3
25
63
200
10
1
25
23
8

Process finished with exit code 0
```

ПРИМЕР

```
print("Введите слово 'stop' для получения результата")
summa = 0
while True:
    x = input("Введите число: ")
    if x == "stop":
        break # Выход из цикла
    x = int(x) # Преобразуем строку в число
    summa += x
print("Сумма чисел равна:", summa)
input ()
```

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: F:\Python\Лекция 1\pr30_2.py ==============
Введите слово 'stop' для получения результата
Введите число: 3
Введите число: 5
Введите число: 4
Введите число: stop
Сумма чисел равна: 12
```

ФАКТОРИАЛ ЧИСЛА

```
n = int ( input())
i =2
p =1
while i <= n:
    p = p * I
    i = i +1
print (p)
```

Пример использования цикла

- Задача Фибоначчи – ряд Фибоначчи – это числовой ряд, каждое следующее значение равно сумме двух предыдущих

- Попробуйте написать программу

ЦИКЛ ОБХОДА ПОСЛЕДОВАТЕЛЬНОСТИ (FOR)

- Цикл while не единственный способ организации повторения группы выражений
- Широко применяется цикл for, который представляет собой цикл обхода заданного множества элементов (символов строки, объектов списка или словаря) и выполнения в своем теле различных операций над ними

ЦИКЛ FOR

```
print(1)
print(2)
.    .    .
print(100)
```

```
for x in range (1, 101): print (x)
```

ЦИКЛ FOR

- Цикл for применяется для перебора элементов последовательности и имеет такой формат:

```
for <Текущий элемент>  in <Последовательность>:
    <Инструкции внутри цикла>
[else:
    <Блок, выполняемый, если не использовался оператор break>
]
```

```
for s in "str":
    print(s, end=" ")
else:
    print("\nЦикл выполнен")
```

ОПЕРАТОР CONTINUE

```
>>> for i in range(1,101):
        if 4<i<11:
            continue
        print(i)


1
2
3
4
11
12
```

ОПЕРАТОР BREAK
ДОСРОЧНОЕ ЗАВЕРШЕНИЕ ЦИКЛА

```
while True:
        x=input('Password=')
        if x=="P@ssw0rd":
                break
```

```
Password=пароль
Password=P@ssw0rd
>>>
```

ЦИКЛ FOR

- Функция range() имеет следующий формат:

```
range([<Начало>, ]<Конец>[, <Шаг>])
```

```
arr = [1, 2, 3]
for i in range(len(arr)):
    arr[i] *= 2
print(arr)
```

ЦИКЛ FOR

```
for x in [1, 2, 3]:
    print(x)
for y in (1, 2, 3):
    print(y)
```

```
arr = [1, 2, 3]
for i in arr:
    i = i * 2
print(arr)
```

```
===================== RESTART: F:\Python\Лекция 1\pr26_2.py ====================
1
2
3
1
2
3
>>>
```

```
===================== RESTART: F:\Python\Лекция 1\pr27_1.py ====================
[1, 2, 3]
>>>
```

- В языке программирования Python цикл for имеет зачастую несколько иное применение
- Список в Python относится к итерируемым объектам, это значит, что его элементы можно обойти циклом for, причём переменная-счётчик будет на каждом шаге принимать значение очередного элемента цикла:

```
lst = [12 , 17.9 , True , -8, False ]
for j in lst :
 print (j )
```

НАПИШЕМ ПРОГРАММУ

- Накопление суммы и произведения 1!+2!+3!+…+n!

С ПОМОЩЬЮ ЦИКЛА FOR МОЖНО ПЕРЕБИРАТЬ СТРОКИ

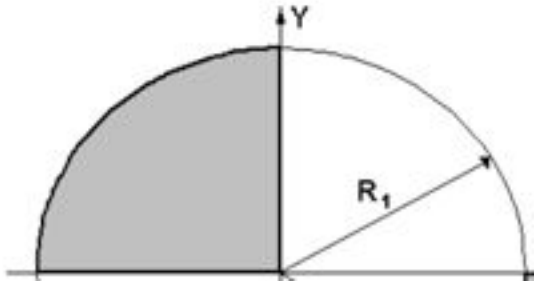
```
str1 = "Привет"
for i in str1 :
    print (i, end = " ")
```

ЗАДАНИЕ

- Создайте цикл, который печатает четные числа до тех пор, пока не выведет ваш возраст. Если ваш возраст — нечетное число, создайте цикл, который печатает нечетные числа до совпадения с возрастом

ДЗ

- Для целого числа k от 1 до 99 напечатать фразу «Мне k лет», учитывая при этом, что при некоторых значениях k слово «лет» надо заменить на слово «год» или «года». Например, 11 лет, 22 года, 51 год.
- Проверка принадлежности диапазону



- Напишите программу, которая запрашивает у пользователя числа до тех пор, пока каждое следующее число больше предыдущего. В конце программа сообщает, сколько чисел было введено