Python

2

Формы оператора присваивания

```
spam = 'Spam'      # Каноническая форма
spam, ham = 'yum', 'YUM'      # Присваивание кортежей (позиционное)
[spam, ham] = ['yum', 'YUM']  # Присваивание списков (позиционное)
a, b, c, d = 'spam' # Присваивание последовательностей, обобщенное
a, *b = 'spam'                # Расширенная операция распаковывания
# последовательностей
spam = ham = 'lunch'     # Групповое присвоение одного значения
spams += 42         # Комбинированная инструкция присваивания
# (эквивалентно инструкции spams = spams + 42)
```

Позиционное присваивание кортежей и списков. Общая форма. Принцип выполнения позиционного присваивания

В простейшем случае общая форма позиционного оператора присваивания кортежей имеет вид:

```
name1, name2, ..., nameN = value1, value2, ..., valueN
```

где

- *name1*, *name2*, *nameN* – имена (переменные) которые нужно связать с объектами (значениями) *value1*, *value2*, *valueN*;
- *value1*, *value2*, *valueN* – значения (объекты), которые связываются с именами *name1*, *name2*, *nameN*.

Позиционное присваивание последовательностей символов

Оператор присваивания можно применять для присваивания последовательностей символов. Среди нескольких имен, символ в правой части присваивается имени, позиция которого в левой части совпадает с позицией этого символа.

```
>>> x,y,z,w = 'abcd'
>>> x
'a'
>>> y
'b'
>>> z
'c'
>>> w
'd'
```

Примеры позиционного присваивания кортежей для обмена их значениями

```
>>> a,b = 5,7 # позиционное присваивание кортежей
>>> a,b
(5, 7)
>>> a,b = b,a # обмен значениями
>>> a,b
(7, 5)
 >>>
```

Позиционное присваивание списков

```
>>> [X,Y,Z] = [5,10,15]  # позиционное присваивание списков
>>> [X,Y,Z]
[5, 10, 15]
>>> X
5
>>> Y,Z
(10, 15)
>>> [Y,Z] # представить как список
[10, 15]
>>> (X,Y,Z) # представить как кортеж
(5, 10, 15)
```

Пример присваивания между кортежами и списками

```
>>> [x, y, z] = (5, 10, 15)  # кортеж значений присваивается списку переменных
>>> x,y,z
(5, 10, 15)
>>> (i,j,k) = [100, 200, 300]  # список значений присваивается кортежу
>>> i,j,k
(100, 200, 300)
>>> (a,b,c,d) = "INFO" # строка символов присваивается кортежу
>>> a,b,c,d
('I', 'N', 'F', 'O')
>>> a,d
('I', 'O')
```

Форматирование строк

```
firstname = input('Input your firstname: ')
lastname = input('Input your lastname: ')


string_a = 'Hello, %s %s' % (firstname, lastname)
string_b = 'Hello, {} {}'.format(firstname, lastname)
string_c = 'Hello, {0} {1}'.format(firstname, lastname)
string_d = f'Hello, {firstname} {lastname}'
```

Форматирование строк с помощью метода format

Если для подстановки требуется только один аргумент, то значение - сам аргумент:

```
>>> 'Hello, {}!'.format('Vasya')
'Hello, Vasya!'
```

А если несколько, то значениями будут являться все аргументы со строками подстановки (обычных или именованных):

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'
>>> '{2}, {1}, {0}'.format(*'abc')
'c, b, a'
>>> '{0}{1}{0}'.format('abra', 'cad')
'abracadabra'
```

Форматирование строк с помощью оператора %

Если для подстановки требуется только один аргумент, то значение - сам аргумент:

```
>>>
>>> 'Hello, %s!' % 'Vasya'

'Hello, Vasya!'
```

А если несколько, то значением будет являться кортеж со строками подстановки:

```
>>>
>>> '%d %s, %d %s' % (6, 'bananas', 10, 'lemons')

'6 bananas, 10 lemons'
```

'%d', '%i', '%u'	Десятичное число.
'%o'	Число в восьмеричной системе счисления.
'%x'	Число в шестнадцатеричной системе счисления (буквы в нижнем регистре).
'%X'	Число в шестнадцатеричной системе счисления (буквы в верхнем регистре).
'%e'	Число с плавающей точкой с экспонентой (экспонента в нижнем регистре).
'%E'	Число с плавающей точкой с экспонентой (экспонента в верхнем регистре).
'%f', '%F'	Число с плавающей точкой (обычный формат).
'%g'	Число с плавающей точкой. с экспонентой (экспонента в нижнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'%G'	Число с плавающей точкой. с экспонентой (экспонента в верхнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'%c'	Символ (строка из одного символа или число - код символа).
'%r'	Строка (литерал python).
'%s'	Строка (как обычно воспринимается пользователем).
'%%'	Знак '%'.
'%d', '%i', '%u'	Десятичное число.

Форматирование строк с помощью f-строк

```
In [1]: ip = '10.1.1.1'

In [2]: mask = 24

In [3]: f"IP: {ip}, mask: {mask}"

Out[3]: 'IP: 10.1.1.1, mask: 24'
```

Аналогичный результат с format можно получить так:

``"IP: {ip}, mask: {mask}".format(ip=ip, mask=mask)``.

Дополнительные функции по работе со строками

```
string = 'Hello,_my_name_is_Alex'
string_list = string.split('_')
result = ' '.join(string_list)


string = 'ping pong'
result_string = string.replace('p', 'k')
```

Структурное программирование

Структурное программирование — методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков.

- Последовательность — однократное выполнение операций в том порядке, в котором они записаны в тексте программы.
- Ветвление — однократное выполнение одной из двух или более операций, в зависимости от выполнения заданного условия.
- Цикл — многократное исполнение одной и той же операции до тех пор, пока выполняется заданное условие (условие продолжения цикла).

Логические выражения

Логическими (boolean expression) называются выражения, которые могут принимать одно из двух значений – истина или ложь. В следующем примере используется оператор ==, который сравнивает два операнда и возвращает True, если они равны (equal) или в противном случае False:

```
>>> 5 == 5
True
>>> 5 == 6
False
```

True и False – специальные значения, которые принадлежат к типу bool

Операторы сравнения

a равно 10 и b равно 20

==	Если значения двух операнд равны - возвращает True	(a == b) - Ложь
!=	Если значения двух операнд **НЕ** равны - возвращает True	(a != b) - Истина
>	Если значение левого операнда больше значения правого - возвращает True	(a > b) - Ложь
<	Если значение правого операнда больше значения левого - возвращает True	(a < b) - Истина
>=	Если значение левого операнда больше, либо равно значению правого - возвращает True	(a >= b) - Ложь
<=	Если значение правого операнда больше, либо равно значению левого - возвращает True	(a <= b) - Истина
is	Если левый и правый операнды ссылаются на один и тот же участок памяти - возвращает True	(a is b) - Ложь
is not	Если левый и правый операнды **НЕ** ссылаются на один и тот же участок памяти - возвращает True	(a is not b) - Правда

Логические операторы

Существуют три логических оператора (logical operators): **and**, **or** и **not**. Смысл этих операторов схож с их смыслом в английском языке:

x > 0 and x < 10
это истинно в случае, если x больше 0 и меньше 10.

n%2 == 0 or n%3 == 0
это истинно, если одно из выражений является истинным.

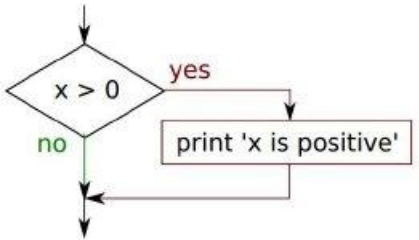
Оператор **not** отрицает логическое выражение, так
not (x > y)
истинно, если x > y является ложью, т.е. x меньше или равно y.

Условное исполнение

Для того чтобы писать полезные программы, почти всегда необходимо проверять условия и изменять поведение программы. Условные инструкции (conditional statements) предоставляют нам такую возможность. Простейшей формой является инструкция if:

```
if x > 0 :
    print('x is positive')
```

Логическое выражение после инструкции if называется условием (condition). Далее следует символ двоеточия (:) и строка (строки) с отступом. Если логическое условие истинно, тогда управление получает выражение, записанное с отступами, иначе – выражение пропускается.



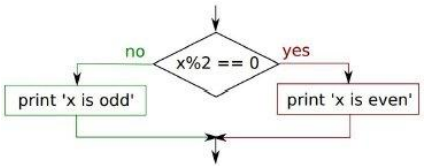
Не существует ограничения на число инструкций, которые могут встречаться в теле if, но хотя бы одна инструкция там должна быть. Иногда полезно иметь тело if без инструкций (обычно оставляют место для кода, который ещё не написан). В этом случае можно воспользоваться инструкцией **pass**, которая ничего не делает.

```
if x < 0 :
   pass # необходимо обработать отрицательные значения!
```

Альтернативное исполнение

Второй формой инструкции if является альтернативное исполнение (alternative execution), в котором существуют два направления выполнения и условие определяет, какое из них выполнится. Синтаксис выглядит следующим образом:

```
if x%2 == 0 :
        print('x is even')
else :
        print('x is odd')
```



Если остаток от деления x на 2 равен 0, то x-четное, и программа выводит сообщение об этом. Если условие ложно, то выполняется второй набор инструкций.

Так как условие может быть либо истинным, либо ложным, тогда точно выполнится один из вариантов. Варианты называются ветвями (branches), потому что они являются ответвлениями в потоке исполнения.

```
tovar1 = 50
tovar2 = 32
if tovar1+ tovar2 > 99 :
    print("99 рублей недостаточно")
else:
    print("Чек оплачен")
```

```
if a > 0 and a < b:
    print(b - a)
```

Чтобы вложенный код выполнился, `a` должно быть больше нуля и одновременно меньше `b`. Также в Питоне, в отличие от других языков программирования, позволительна такая сокращенная запись сложного логического выражения:

```
if 0 < a < b:
    print(b - a)
```

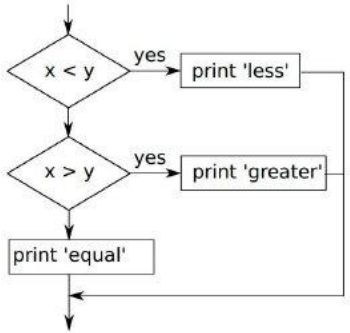
Оператор in

```
string = 'Hello'
if 'e' in string:
    print('YES')
else:
    print('NO')
```

Последовательность условий

Иногда имеется больше двух вариантов выполнения, тогда нам необходимо больше двух ветвей. В этом случае, можно воспользоваться сцепленными условиями (chained conditional):

```
if x < y:
    print('less')
elif x > y:
    print('greater')
else:
    print('equal')
```



elif является аббревиатурой от "else if". Будет исполнена точно одна ветвь.

Не существует ограничения на количество инструкций elif. Если встречается оператор else, то он должен быть в конце, но может не быть ни одного.

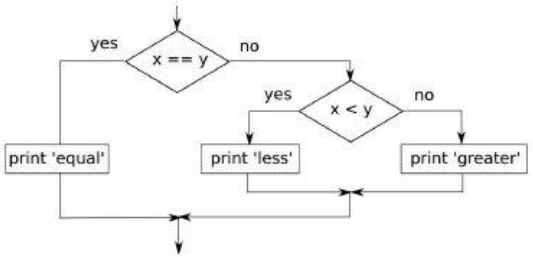
```
if choice == 4:
    print('Bad guess')
elif choice == 5:
   print('Good guess')
elif choice == 8:
   print('Close, but not correct')
```

Каждое условие проверяется в порядке расположения. Если первое условие ложно, то проверяется следующее и т.д. Если одно из условий истинно, то выполняется соответствующая ветка, и инструкция завершается. Даже если верно более чем одно условие, все равно выполняется только первая истинная ветка.

Вложенные условия

Одно условие может быть вложено в другое. Мы можем записать пример трихотомии (trichotomy):

```
if x == y:
    print('equal')
else:
    if x < y:
        print('less')
    else:
        print('greater')
```



Внешнее условие содержит две ветки. Первая ветка содержит простую инструкцию. Вторая ветка содержит еще одну инструкцию if, которая имеет две ветки. Эти две ветки являются простыми инструкциями, хотя они также могут содержать инструкции.

Логические операторы часто позволяют упростить вложенные условные инструкции. Например, мы можем записать следующий код, используя одно условие:

```
if x > 0:
    if x < 10:
        print ('x is a positive single-digit number.')
```

Инструкция print выполнится только, если мы зададим ее после обоих условий, также мы получим похожий эффект, используя оператор and:

```
if x > 0 or x < 10:
    print ('x is a positive single-digit number.')
```

Условные операторы

Конструкция if

```
If выражение:
    инструкция
```

Конструкция if-else

```
If выражение:
    инструкция1
else:
    инструкция2
```

Конструкция if-elif-else

```
If выражение1:
    инструкция1
elif выражение2:
    инструкция2
else:
    инструкция3
```

```
age = int(input('--> '))
```

Конструкция if

```
If age >= 18:
    print(age)
```

Конструкция if-else

```
If age >= 18:
    print('Yes')
else:
    print('No')
```

Конструкция if-elif-else

```
If age >= 18:
    print('beer')
elif age <= 12:
    print('candy')
else:
    print('orange')
```

Задание 1

Напишите программу , которая просит пользователя что-нибудь ввести с клавиатуры. Если он вводит какие-нибудь данные, то на экране должно выводиться сообщение "ОК". Если он не вводит данные, а просто нажимает Enter, то программа ничего не выводит на экран.

Задание 2

Напишите программу , которая запрашивает у пользователя число. Если оно больше нуля, то в ответ на экран выводится число 1. Если введенное число не является положительным, то на экран должно выводиться -1.