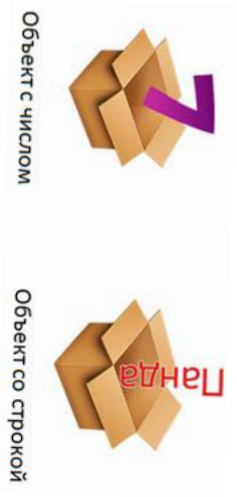
Переменные, оператор присваивания, функции type и id

Вначале, давайте попробуем понять, что вообще могут делать компьютерные программы? В самой основе, совсем немного:

- хранить данные;
- выполнять арифметические операции;
- проверять логические условия (операторы ветвления);
- реализовывать циклы.

Как же данные представляются в Python?



Условно, это можно представить так. Есть некое хранилище (мы его будем называть объектом) и в нем могут располагаться или числа, или строки, или, какие-либо другие типы данных.

Причем в объекте может быть что-то одно: или число, или строка. Одновременно и число и строка находиться в одном хранилище не могут.

Переменные и оператор присваивания

Таких хранилищ в программе может быть огромное количество. И как нам тогда обратиться к нужному и взять оттуда данные? Все просто. Для этого у нас должна быть ссылка на объект и мы обращаемся к хранилищу по имени этой ссылки. Такие ссылки называются Переменные

Как нам создать объект с некоторым содержимым и ссылкой на него?Тоже очень просто. Достаточно придумать имя переменной и через оператор присваивания связать ее с нужным объектом, например, так:

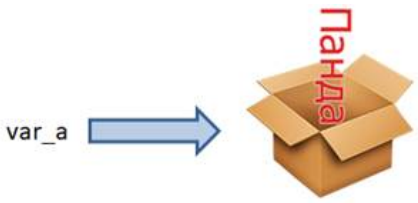
```
a = 7
```

В результате, интерпретатор языка Python создаст ссылку с именем a и объектом с целым числом 7. Мы в этом можем легко убедиться, если выведем содержимое объекта по этой ссылке (по переменной a):

```
print(a)
```

Например, строчка:

```
var_a = "Панда"
```



Еще раз обратите внимание на оператор присваивания. В программировании он связывает операнд слева с операндом справа:

операнд слева = операнд справа

создает переменную с именем var_a, которая ссылается на объект со строкой «Панда». И, смотрите, здесь именно создается переменная var_a, так как до этого она не существовала. К несуществующим переменным мы обращаться не можем. Например, строчка:

```
print(x)
```

приведет к ошибке, так как переменную с именем x мы нигде не создавали. Но если переменная уже была создана и мы снова присваиваем ей какое-либо значение:

```
a = 6.8
```

то она второй раз уже не создается. Но как будет работать эта строчка? Здесь создается новый объект со значением 6.8 и на него инициализируется уже существующая переменная. Если на прежний объект 7 нет других ссылок, то он автоматически удаляется из памяти устройства. При этом переменная a начинает ссылаться на другой тип данных – вещественное число (до этого было целочисленное значение). То есть, тип переменной определяется в момент присваивания ей того или иного значения. В

программировании это называется **динамической типизацией**. В противовес строгой типизации, когда тип переменной указывается в момент ее объявления. Например, так делается в языках C++ или Java

Но, вернемся к оператору присваивания. Давайте посмотрим, что будет, если связать одну переменную с другой:

```
b = a
```

Теперь обе переменные будут ссылаться на один и тот же объект. То есть, копирования данных (создание нового объекта) не происходит, копируется лишь ссылка на объект. И это важный момент, который хорошо следует запомнить и понимать! В языке Python переменные не хранят значения, а лишь ссылаются на них. Как раз, благодаря этому можно одной и той же переменной присваивать самые разные типы данных:

```
b = "Hello"
b = 0
b = -8.4
```

Это очень удобно при программировании. Но при этом всегда следует помнить, что переменная – это всего лишь ссылка на данные, а не сами данные.

Иногда, в программах можно встретить вариант **каскадного присваивания**:

```
a = b = c = 0
```

В результате выполнения такой команды, все три переменные будут ссылаться на один и тот же объект со значением 0.

Если же мы хотим, чтобы каждая переменная ссылалась на свой отдельный объект, то можно воспользоваться **множественным присваиванием**:

```
a, b = 1, 2
```

Здесь переменная a будет ссылаться на 1, а b – на 2. Используя такую команду, можно легко и просто выполнить операцию обмена значениями между двумя переменными:

```
a, b = b, a
```

Функция type()

Так как в программе переменные могут иметь самые разные типы данных, то как можно узнать текущий тип, на который они ссылаются? Для этого в Python имеется специальная встроенная функция type(), возвращающая тип данных, связанный с указанной переменной:

```
print(type(a))
```

В консоли увидим целочисленный тип данных (int). Давайте объявим еще две переменные:

```
x = 5.8
s = "Hello"
```

И выведем для них типы:

```
print(type(x), type(s))
```

Увидим float и str. Здесь у вас может возникнуть вопрос, а какие типы данных вообще существуют в Python? Конечно, я бы мог здесь привести список всех типов, но на данном этапе – это избыточная, справочная информация. По мере прохождения курса, мы с вами познакомимся со всеми встроенными типами и постепенно у вас сложится общее представление. Это будет гораздо эффективнее, чем отображение больших и умных таблиц, которые, все равно, не воспринимаются и быстро забываются.

Правильные имена переменных

Последнее, о чем я хочу вам рассказать на этом занятии – как правильно выбирать имена переменных. Есть несколько простых правил:

1. Имена следует брать существительными (отвечают на вопросы: кто, что).
2. Имена должны быть осмысленными и отражать суть данных.
3. Допустимые символы в именах: первый символ – любая буква латинского алфавита a-z, A-Z и символ подчеркивания _. В качестве второго и последующих символов еще цифры 0-9.

Например:

```
msg = "Сообщение"
count = 0
```

Причем, обратите внимание, переменные:

```
arg = 0
Arg = 0
```

Это две разные переменные, так как малая буква ‘a’ и большая ‘A’ – разные символы, а значит, имена тоже разные. Также нельзя использовать ключевые слова языка Python в качестве имен, например, писать:

```
True = 5
```

Полный их список можно посмотреть с помощью вызова в консоли функции:

help()

а, затем, набрать:

keywords

Также не следует использовать имена стандартных функций в качестве переменных. Например, если переопределить имя функции:

```
print = 5
```

то оно теперь будет ссылаться не на функцию для печати значений, а на числовой объект 5. И если, затем, попытаться вызвать функцию:

```
print(6)
```

то возникнет ошибка, так как мы, фактически, пытаемся выполнить объект, содержащий число 5. Но Python делать этого не умеет (по крайней мере, по умолчанию). Поэтому имена функций стандартной библиотеки языка Python не следует использовать в качестве имен переменных.

Но как нам узнать, является ли какое-либо имя встроенной функцией? Очень просто. Все зарезервированные имена, будь то функции или ключевые слова, автоматически подсвечиваются