17. Конспект

Конспект поможет вам вспомнить информацию, которая была в видеоуроке, пригодится для дальнейшего обучения и работы.

Colab

Код на Python выполняется обычно на вашем компьютере или на сервере. Но пока в первое время обучения мы будем использовать **Colab**.
Colab — специальный инструмент от Google, который запускается онлайн, выполняет Python в облаке и показывает нам результат.

Это бесплатный сервис, который вы можете использовать для своих экспериментов. Домашнее задание вы тоже будете делать в нем.
Для работы с Colab вам потребуется гугл-аккаунт.

Перейти в Colab и подробнее почитать об инструменте можно по ссылке: https://colab.research.google.com.

Переменные

Переменная — это ячейка в памяти компьютера, у которой есть имя и в которой хранятся данные.

Чтобы создать переменную, запишем:

```
name = "Сергей"
```

name — **имя** переменной,

"Сергей" — **значение** переменной,

знак равно (=) — **знак присваивания**.

Чтобы обращаться в программном коде к этим данным, мы будем использовать имя переменной — **name**.

Правила написания имен переменных

- Имя переменной может состоять только из букв, цифр и знака подчеркивания.
- Имя не может начинаться с цифры.
- Имя не может содержать специальных символов @, $, % и т. д.
- Имя не может начинаться с пробела.
- Регистр в наименовании переменных смыслоразличительный (name и Name — это две разные переменные).
- Когда имя переменной передает смысл, читать код проще.

Примеры:
```
girls_after_18 # понятное имя переменной
a1 # непонятное имя переменной
1st-broken var # нерабочее имя переменной
```

Чтобы вывести информацию из переменной на экран, передадим переменную **name** в функцию print():
```
name = "Сергей"
print(name)
>>> Сергей
```

Функция print(). Операции сложения, вычитания, умножения и деления

Рассмотрим, что такое функция, и разберемся подробнее, что делает функция print().

Функция — это именованная часть кода, которая выполняет определенную задачу.

То, что мы передаем в функцию внутри скобок, называется **аргументом**.

Функция записывается следующим образом:
```
имя_функции(аргументы_через_запятую)
```

Пример функции **print()** с переменной age:

```
age = 25
print(age)
>>> 25
```

Функция print() выводит данные на экран, т. е. печатает на отдельной строке.

print() позволяет:

- Вывести пустую строку:

  ```
  print()
  ```

- Вывести текст:

  ```
  print("Доброе утро")
  ```

  ```
  >>> Доброе утро
  ```

- Использовать математические операторы — сложение (+), вычитание (–), деление (/), умножение (*):

  ```
  print(80+40+50)
  ```

  ```
  >>> 170
  ```

- Комбинировать текст, числовые значения и арифметические операции:
- `print("Кофе —-", 80)`
- `print("Булочка —-", 40)`

  ```
  print("Сумма вашей покупки —-", 80+40)
  >>> Кофе —- 80
  >>> Булочка —- 40

  >>> Сумма вашей покупки —- 120
  ```

- Вывести значение переменной:
- `name = "Сергей"`

  ```
  print(name)
  ```

  ```
  >>> Сергей
  ```

Обратите внимание, **переменные пишутся без кавычек**. Если указать их имена в кавычках, будут выведены просто строковые значения.

- Передавать несколько значений через запятую:

```
sugar = 3 # ложки
eggs = 1 # штуки
flour = 3 # столовые ложки
apples = 1 # штуки
print("Сахар", sugar , "ст. л.")
print("Яйца", eggs , "шт.")
print("Мука", flour , "ст. л.")

print("Яблоки", apples, "шт.")
```

- Передаваться в другую функцию:

```
print(int("2"))

>>> 2
```

О функции int() подробнее рассказано в разделе «Преобразование типов данных».

Дополнительно о функции print()

Типы данных

Переменные могут хранить разную информацию — текст или числовое значение. У переменной, кроме имени и значения, есть еще одна сущность — тип.

На уроке мы изучили три типа данных: строка, целое число и число с плавающей точкой.

Строка

str (string) — строка, т. е. набор символов.

Строки записываются в одинарных или двойных кавычках:
```
my_string = 'одинарные кавычки для строк'
my_string = "двойные кавычки для строк"
```

Строки — это:

- все системные сообщения;
- логины и пароли;
- почта;
- названия городов, стран, улиц;
- название приложений, модулей, моделей телефонов и т. д.

Операции, которые используются со строками:

- сложение — добавит одну строку к другой,
- умножение — повторит строку n раз.

Чтобы при сложении между строками был пробел, его нужно добавить в выражение.

Рассмотрим операции сложения и умножения на примере:
```
name = 'Антон'
surname = 'Васильев'
print(name * 3)
print(name + ' ' + surname)
```

Получим:
```
АнтонАнтонАнтон
Антон Васильев
```

Это пригодится

Если нужно разместить строку на нескольких строчках, то используем тройные (''') кавычки.

Пример:
```
my_string = '''используем тройные кавычки,
```

```
чтобы разместить текст
на нескольких строчках
для удобства чтения кода
'''
```

Целые числа

int (integer) — целые числа. Например, 1, 5, 10, 100 и т. д.

Примеры:
```
age = 25 # возраст
height = 192 # рост
```

Числовой тип данных используется для математических действий.

Обратите внимание, что числа записываются без кавычек. Если записать число в кавычках, то получится строка.

Дробные числа

float — дробное число, или число с плавающей точкой.

Примеры:
```
tower_height = 28.365 # высота башни
temperature = 36.6 # температура
```

Функция type()

Если вы не знаете, какой у вас тип данных, то используйте функцию type().

Функция type() показывает тип данных переменной, которую мы ей передаем.

Записывается так:
```
type(имя_переменной)
```

Пример:
```
a = 'неизвестный тип данных'
b = 1
print(type(a))
print(type(b))
```

Результат работы программы:
```
<class 'str'>
<class 'int'>
```

Функция ввода input()

Дополняет функцию print().

Функция input() записывает введенное пользователем значение

в переменную.

Пример:
```
name = input()
print("Вы ввели", name)
```

Пользователь вводит имя, например Сергей. Программа выводит:
```
Вы ввели Сергей
```

Чтобы сообщить пользователю, что нужно ввести, запишем текст внутри

скобок функции **input()** в качестве аргумента:
```
phone_model = input("Какой смартфон вы хотите купить?")
print("Вы хотите купить", phone_model)
```

Пользователь сначала видит вопрос с полем ввода:
```
Какой смартфон вы хотите купить?
```

После того как пользователь вводит информацию, например iPhone,

программа выводит:
```
Вы хотите купить iPhone
```

Обратите внимание, что функция **input()** передает в программу **строку**.

Проверим это с помощью функции type():
```
price = input("Сколько стоит эта модель?")
price_type = type(price)
print(price_type)
```

Получим:
```
<class 'str'>
```

Если мы забудем, что тип полученных данных от пользователя — это

строка, то можем допустить ошибку. Посчитаем стоимость модели

телефона с кредитом:
```
price = input("Сколько стоит эта модель?")
total_price = price*2
print("Полная стоимость с кредитом:", total_price)
```

Введем стоимость модели — 10000. Получим такое сообщение:
`Полная стоимость с кредитом: 1000010000`

Давайте разберемся, как избежать подобных ошибок.

Преобразование типов данных

Чтобы работать с данными, которые вводит пользователь, воспользуемся функциями приведения типов:

- **str()** — приводит число к строке.
- `price = 34.5`
- `my_str = "Цена мороженого " + str(price)`

  `print(my_str)`

  `>>> Цена мороженого 34.5`

- **int()** — приводит строку или дробное число к целому числу.
- `a = int("68")*3`
- `b = int(33.8) + 29`
- `c = a + b`

  `print(c)`

  `>>> 266`

  Обратите внимание, что функция **int()** не округляет дробное число (33.8 >> 34), а отбрасывает знаки после запятой (33.8 >> 33).

  Если в функцию **int()** записать текст, то такой код приведет к ошибке:

  `int("текст")`

- **float()** — приводит строку или целое число к дробному числу.
- `a = float("68.5")*3`
- `b = float(33) + 29.4`
- `c = a + b`

  `print(c)`

  `>>> 267.9`

В одном выражении можно использовать несколько функций приведения типов.

В этом примере данные сначала будут приведены к целому числу, затем к строке:

```
str(int())
```

Работаем с данными, которые ввел пользователь

Давайте теперь разберемся, как получить от пользователя числовые данные. Вернемся к задаче со смартфоном.

Получим от пользователя данные с помощью функции **input()**, а затем преобразуем в число с помощью функции **int()** или **float()**:

```
price_str = input("Стоимость смартфона")
price_int = int(price_str)
```

Мы можем сразу в функцию **int()** или **float()** передать **input()** — тогда то, что введет пользователь, будет преобразовано в число, если это возможно:

```
year = int(input("Год выпуска"))
money = float(input("Денег в наличии"))
```

Исправим задачу со смартфоном:

```
price = input("Сколько стоит эта модель? ")
price_int = int(price)
total_price = price_int*2
print("Полная стоимость с кредитом:", total_price)
```

Введем 10000 и проверим работу кода:

```
20000
```

Типизация в Python

Частый вопрос на собеседовании: «Какая типизация у языка Python?»

Давайте разберемся, что такое типизация языков программирования и какие виды типизации бывают.

Типизация — это то, как язык программирования распознает типы переменных.

То есть типизация определяет, как работать с переменными: нужно ли задавать тип переменных или язык определит его сам, можно ли изменять тип и т. д.

Типизация бывает:

- динамическая и статическая,
- сильная (строгая) и слабая (нестрогая).

Динамическая и статическая типизация

В языках со **статической типизацией** программист прописывает типы переменных.

В примере кода статически типизированного языка C++ обязательно указывается тип переменных, в данном случае — int:

```
int a = 8;
int b = 2;
int result;

int sum(int a,int b) {
      return (a+b);
}

result = sum(a, b);
```

В **динамически типизированных языках** мы не пишем, какого типа переменная. Тип определяется во время выполнения программы — динамически.

Пример кода динамически типизированного языка Python:

```
a = 8
b = 2

def sum(a, b):
    return a + b

result = sum(a, b)
```

В процессе работы программы Python вычислит значение выражения справа от знака равно и определит исходя из значения, какого типа полученное значение.

В первой строке кода Python определит тип полученного значения **8** как **int**. Значит, переменная **a** будет типа **int**.

Динамически типизированные языки программирования: Python, JavaScript, PHP, Perl, Ruby, Lisp.

Статически типизированные языки программирования: Java, C#, C, C++, Go, Pascal, Scala, Rust.

Сильная и слабая типизация

Если в **языках со слабой (нестрогой) типизацией** прописать действия с разными типами данных, обработка которых не регламентирована на уровне языка, то компилятор будет пробовать делать неявное преобразование типов и выполнять эту операцию.

Рассмотрим на примере кода языка со слабой типизацией — JavaScript.

Здесь компилятор приведет 5 к строке и суммирует строки:

```
5 + '6';
```

Мы получим строку:

```
'56'
```

В примере ниже строка '3' будет приведена к типу int, а затем 4 будет умножено на 3:

```
4 * '3';
>>> 12
```

В языках **сильной (строгой) типизации** важно — прописаны или не прописаны правила для проведения какой-либо операции с типами.

Рассмотрим пример сильной (строгой) типизации на языке Python. Если мы сложим число 5 и строку '6', то получим ошибку:

```
5 + '6'
```

В языках сильной (строгой) типизации не делается неявное преобразование типов.

Код ниже выполнится, потому что в Python можно умножать строку на число:

```
'6'*5
>>> '66666'
```

Языки с сильной типизацией: Python, Java, C#, Ruby, Go, Pascal, Scala, Rust, Lisp.

Языки со слабой типизацией: JavaScript, C, C++, PHP, Perl.

Python — язык сильной динамической типизации.

f-строки

Вы уже умеете соединять строки с помощью знака плюс (+). Это базовый способ соединения строк — **конкатенация**.

Конкатенация — операция, которая «суммирует» строки друг с другом.

Пример:

```
first_name = 'Skypro'
greeting = 'Hello'
print(greeting + ", " + first_name + "!")
>>> Hello, Skypro!
```

Если строка устроена сложно, то читать и увидеть конечный результат в такой записи тяжело.

Есть другой способ соединения строк — **интерполяция**. Для этого мы используем **f-строки**, где в фигурных скобках { } записываем имена переменных:

```
first_name = 'Skypro'
greeting = 'Hello'
```

```
print(f'{greeting}, {first_name}!')
```

Мы создали **f-строку** — шаблон, в который с помощью фигурных скобок подставляются значения переменных.

На выходе получается обычная строка:
```
Hello, Skypro!
```

Функция **round()** округляет число с плавающей точкой до знака, который вы указываете.

Записывается так:
```
round(<число>, <количество знаков после точки>)
```

Между числом и количеством знаков ставится **запятая**. Дробное число записывается с помощью **точки**.

Пример:
```
round(5.7777,2)
>>> 5.78
```

Обратите внимание, если мы не указываем количество знаков после точки, то по умолчанию это будет 0. Число округляется до целого и возвращается целое число.

Пример:
```
round(5.7777)
>>> 6
```