ПАРАДИГМЫ.

Существует три основных парадигмы: **структурное, объектно-ориентированное и функциональное**. Интересно, что сначала было открыто функциональное, потом объектно-ориентированное, и только потом структурное программирование, но применяться повсеместно на практике они стали в обратном порядке.

ПЕРЕМЕННЫЕ.

Переменная в языке программирования это название для зарезервированного места в памяти компьютера, предназначенное для хранения значений. Это означает, что когда вы создаете переменную, вы на самом деле резервируете определенное место в памяти компьютера.

Основываясь на типе данных переменной, интерпретатор выделяет необходимое количество памяти и решает, что может находится в зарезервированной области памяти.

Для понимания, можете думать о переменной как о коробке, в которую можно положить любую вещь, но только определенного размера. Размер в данном примере будет типом переменной. Это не совсем верное определение, но оно дает общее представление о картине в целом.

**Объявление переменной:**

Имя\_переменной = значение\_переменной

ОПЕРАТОРЫ.

1. Арифметические операторы (+, -, \*, /, //, %, \*\*);
2. Операторы сравнения (реляционные - <, <=, ==, !=, >, >=);
3. Операторы присваивания (=);
4. Побитовые операторы (&, |, ^, ~, <<, >>);
5. Логические операторы (and, or, not);
6. Операторы членства - Операторы членства используются для определения того, лежат ли переменные в последовательности (список, кортеж, строка и словарь) или нет (in, not in);
7. Операторы тождественности - проверяют, находятся ли два значения (или две переменные) по одному адресу в памяти. То, что две переменные равны еще не значит, что они идентичны (is, is not).

ЛОГИЧЕСИКИЕ ВЫРАЖЕНИЯ.

1. == - логический оператор равно, который возвращает в ответ истину(True) или ложь(False). Пример: 2\*2 == 4(True), 2\*3 == 4(False);
2. < - логический оператор меньше. Пример: 2 < 3(True), 3<1(False);
3. > - логический оператор больше. Пример: 4 > 3(True), 3 > 5(False);
4. >= - логический оператор больше или равно. Пример: 4 >= 4(True), 3 >= 5(False);
5. <= - логический оператор меньше или равно. Пример: 4 <= 4(True), 3 <= 0(False);
6. != - логический оператор не равно. Пример: 2\*3 != 4(True), 2\*2 != 4(False);

ЦИКЛЫ.

В Python циклы работают так же, как в большинстве других языков программирования. Программа повторяет определённое действие несколько раз. Действие задано в теле цикла, число повторений — в его условии. В условии может быть вполне конкретное число, например, пять или десять. А может быть более сложная проверка — например, анализ переменной из тела цикла.

1. Цикл for в Python используется **для перебора последовательностей** (списков, кортежей, строк) и других итерируемых объектов;
2. Цикл while (“пока”) позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно.

ТИПЫ ДАННЫХ

1. изменяемые (списки, словари и множества);
2. неизменяемые (числа, строки и кортежи);
3. упорядоченные (списки, кортежи, строки и словари);
4. неупорядоченные (множества).
5. Логические(числа);
6. Веторные(строки, кортежи, списки, словари, множества).

ФУНКЦИЯ.

**Функция** — это фрагмент программного кода, который решает какую-либо задачу.

Синтаксис:

Любые функции и методы начинаются с инструкции **def** (от английского define — «определять»). За def в Python идет название функции, после — набор аргументов. Двоеточие в конце открывает блок тела функции, которое располагают с новой строки.

В Python нет фигурных скобок для блока кода. Вместо них применяют отступы в четыре пробела. Новая инструкция — новая строка. Или можно отделить их точкой с запятой. Чтобы завершить функцию и вернуть из нее значение, используют ключевое слово **return**. Если этого не указать, то, когда тело функции достигает конца, функция завершится. Возвращается объект типа **None**.

ООП.

**Объе́ктно-ориенти́рованное программирование (сокр. ООП)** — методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

class Models:

    def A1(self):

        print("- Hello!")

    def A2(self):

        print(" - All - ")

    def A3(self):

        print("normal.")

class Successor(Models):

    def B1(self):

        print(" - So nice to meet you.")

    def B2(self):

        self.A1()

        self.A2()

        self.A3()

        print(" - And you?")

dialogue\_one = Successor()

dialogue\_one.A1()

dialogue\_one.A2()

dialogue\_one.A3()

dialogue\_one.B1()

dialogue\_two = Successor()

dialogue\_two.B2()