3. Переменные в Python. Специфика создания объектов и присвоения в Python.

Переменные. Для доступа к объекту предназначены переменные. **В языке Python все переменные являются ссылками на объекты**. В Python при инициализации в переменной сохраняется ссылка (адрес объекта в памяти компьютера) на объект. Благодаря этой ссылке можно в дальнейшем использовать объект из программы.
В Python для присвоения используется '=':

```
check = "КОТ МЯУ"
```

Для сравнения значений на равенство используется '==':

```
if check == "КОТ МЯУ":
```

Ну и не равно, следовательно, !=:

```
if check != "КОТ":
```

— Первое присвоение значения переменной создает ее.
— Нет необходимости декларировать тип переменной т.к. в Python используется неявная типизация.

Оператор '=' связывает ссылку на объект (переменную) с объектом, находящимся в памяти.
— Если ссылка на объект (переменная) уже существует, ее легко можно связать с другим объектом, указав этот объект справа от оператора '='.
— Если ссылка на объект еще не существует, она будет создана оператором '='.

Проверить, ссылаются ли две переменные на один и тот же объект в памяти, позволяет оператор is:

```
l = [4,5,2]
m = l
print(m is l) #True
m = l.copy()
print(m is l) #False
```

Для стандартизации записи кода в Python создан стандарт PEP8.
Согласно этому стандарту, **имя переменной** должно удовлетворять следующим условиям:

- Каждая переменная должна иметь уникальное имя, состоящее из латинских букв, цифр и знаков подчеркивания, причем имя переменной не может начинаться с цифры.
- При указании имени переменной важно учитывать регистр букв.
- Следует избегать указания символа подчеркивания в начале имени, т. к. идентификаторы с таким символом имеют специальное значение.
- В качестве имени переменной нельзя использовать ключевые слова.
- Имена переменных должны состоять из маленьких букв, а слова разделяться символами подчеркивания.

ПРОГРАММИСТ ПОМНИ!
В Python изменяемые объекты нельзя скопировать, присвоив одну переменной другой, так как в этом случае копируется ссылка на объект, а не он сам. В итоге при изменении

объекта через одну переменную, изменения видны через другую:

```
check = list(range(5)) # [0, 1, 2, 3, 4]
check1 = check
check1[0] = 10
print(check) #[10, 1, 2, 3, 4]
print(check1) #[10, 1, 2, 3, 4]
```

Поэтому используются иные способы копирования, например, для **поверхностного копирования** могут использоваться следующие способы:

— Срезы:

```
check = list(range(5)) # [0, 1, 2, 3, 4]
check1 = check[:]
check1[0] = 10
print(check) #[0, 1, 2, 3, 4]
print(check1) #[10, 1, 2, 3, 4]
```

— Модуль copy:

```
import copy
check = list(range(5)) # [0, 1, 2, 3, 4]
check1 = copy.copy(check)
check1[0] = 10
print(check) #[0, 1, 2, 3, 4]
print(check1) #[10, 1, 2, 3, 4]
```

— Метод list.copy

```
check = list(range(5)) # [0, 1, 2, 3, 4]
check1 = check.copy()
check1[0] = 10
print(check) #[0, 1, 2, 3, 4]
print(check1) #[10, 1, 2, 3, 4]
```

Если же требуется глубокая копия объекта (полная), то тут только один способ - при помощи метода deepcopy() модуля copy:

```
import copy
check = [list(range(3)) for _ in list(range(2))]
#check = [[0, 1, 2], [0, 1, 2]]
print(check)
check1 = copy.deepcopy(check)
check1[0][0] = 10
print(check) #[[0, 1, 2], [0, 1, 2]]
print(check1) #[[10, 1, 2], [0, 1, 2]]
```

Более понятный ответ на копирование списков есть в билете №9. Копирование списков.