Переменные и типы данных

Переменная - "коробка с наклейкой", в которую можно что-то положить. Например: число или текст, чтобы в дальнейшем его использовать. У коробки есть имя, чтобы можно было узнать, что в ней лежит.

```
name = "Viktor"
age = 21
```

1. `name` и `age` - имена переменных
2. `=` - оператор присвоения, или же способ сказать "положи это в коробку"
3. `"Viktor"` и `21` - содержимое переменных
   Как использовать?

```
print(name)
print(age)
```

`print()` - команда, которая показывает что-то на экране

Правила наименования переменных

1. Имя переменной может содержать буквы, цифры и подчеркивание
2. Имя переменной не может начинаться с цифры
3. Имя переменной чувствительно к регистру. `User` и `user` - не одно и тоже
4. Имя переменной должно отражать ее действительность
5. Нужно использовать стиль `snake_case` : слова разделяются подчеркиванием, все буквы маленькие
6. Нельзя создать переменную с ключевыми словами.
   Ключевые слова представлены ниже
   `False, None, True, and, as, assert, async, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield`
   Если попробовать использовать зарезервированное слово, Python выдаст ошибку (`SyntaxError`)

Типы данных - то, что можно положить в коробку

В переменные можно класть различные виды данных

1. Текст (строки, `str`). Например: `"привет"` или `"http://localhost:3000"`. Текст всегда записывается в кавычки (`" "` или `' '`)
2. Числа:
    1. Целые числа (`int`): `42` , `21` , `0` , `-10`
    2. Дробные числа/числа с запятой (`float`): `42.0` , `0.01` , `3.14`
3. Булевые значения - да/нет (`bool`):
    1. Да - `True`
    2. Нет - `False`
4. Ничего (`None`) - пустая переменная

Примеры инициализации переменных

```
name = "Viktor" # строка с именем
age = 21 # возраст в цифрах
height = 178.8 # рост в дробных числах
man = True # мужской пол в булевом значении
error = None # отсутствие ошибки в "пустой" переменной
```

Условные операторы - принятие решений

Классическая конструкция `if`

Условные операторы выполняют следующую функцию: "Если что-то правда, сделай одно, а если нет - сделай другое". Например, если сервер вернул `200` , скажи `"Все ок"` , а если `404` - выведи `"Ошибка"`

```
statusCode = 200
if statusCode == 200:
    print("Все ок")
```

1. `if` - проверяет условие
2. `==` - значит "равно ли?", является оператором сравнения

3. После `if` ставится `:` и код внутри сдвигается вправо (называется отступ)

```
time = 0.5
if time > 10:
    print("Быстро")
```

Конструкция `if`, `elif`, `else`

Это способ реализовать следующую задумку: "Проверь условие, сделай что-то в зависимости от результата". Простыми словами, это как развилка: делай это, если другое - делай то

1. `if` - проверяет первое условие. Если оно истинное (`True`) - выполняется код внутри
2. `elif` - это сокращение от `else if` (иначе если). Проверяет другое условие, если предыдущее `if` или `elif` не сработало
3. `else` - выполняется, если ни одно из условий выше не истинное

```
status == 200

if status == 200: # Если статус равен 200
    print("OK") # Выведи OK
elif status == 404: # А если статус равен 404
    print("Not found") # Выведи Not found
elif status == 500: # А если статус равен 500
    print("Server error") # Выведи Server error
else: # Если статус не равен ни 200, ни 404, ни 500
    print("Unknown status") # Выведи Unknown status
```

Конструкция match-case - проверка переменной для нескольких случаев

Это "синтаксический сахар" языка, который поможет сделать код боле приятным для просмотра

```
status = 200

# Старт конструкции match-case
# Используется ключевое слово match, после чего идет имя переменной
match status:
    case 200:
        print("OK")
    case 404:
```

```
        print("Not found")
    case 500:
        print("Server error")
    case _: # Аналог оператора else - для всех остальных случаев
        print("Unknown status")
```

Когда лучше использовать `match-case` вместо `if + elif + else`?

1. Когда много условий, и нужно сравнивать одно значение
2. При сложных условиях: `time < 0.5 and status == 200`

Операторы сравнения

Операторы сравнения - специальные символы, которые используются, чтобы сравнить два значения. Результат сравнения всегда либо `True` либо `False`. Эти операторы часто используются в конструкциях `if + elif + else` либо же `match-case`
Список операторов сравнения:

1. `==` - равно
2. `!=` - не равно
3. `<` - меньше
4. `>` - больше
5. `<=` - меньше или равно
6. `>=` - больше или равно

Логические операторы

Логические операторы позволяют комбинировать несколько условий. Они могут работать в связке с операторами сравнения, а также применяются в других случаях.

1. `and` - И
   Оператор проверяет, что оба условия истинны. Если хотя бы одно условие ложно, результат будет `False`.

```
status_code = 200
time = 0.5

if status_code == 200 and time == 0.5:
    print("Success")
else:
    print("Failed")
```

2. `or` - ИЛИ
   Оператор проверяет, что одно ил условий истинно. Если оба условия ложны, результат будет `False`

```
status_code = 200
time = 1
if status_code == 200 or time == 2:
    print("OK") # Выведет OK, т.к. status_code == 200
else:
    print("Failed")
```

3. `not` - НЕ
   Оператор меняет значение на противоположное: `True` становится `False`, а `False` -> `True`

```
error = False
if not is_error: # not is_error -> not false -> true -> OK
    print("OK")
else:
    print("Error")
```
