InnoDom

Базовые конструкции

Операторы сравнения

это операторы, сравнивающие **два** выражения и возвращающие `True` либо `False`, в зависимости от результата

Сравнивать можно логический тип, строки, числа и проверять переменную на пустоту.

Операторы сравнения

Оператор	Описание
==	Проверяет равны ли выражения. Если да, то условие становится истинным.
!=	Проверяет равны ли выражения. Если нет, то условие становится истинным.
>	Проверяет больше ли значение левого выражения, чем значение правого. Если да, то условие становится истинным.
<	Проверяет меньше ли значение левого выражения, чем значение правого. Если да, то условие становится истинным.
>=	Проверяет больше или равно значение левого выражения, чем значение правого. Если да, то условие становится истинным.
<=	Проверяет меньше или равно значение левого выражения, чем значение правого. Если да, то условие становится истинным.

Операторы ветвления

это виды операторов, которые, в зависимости от каких-то явлений, меняют свое поведение

- **Условные операторы**
- **Обработчики исключений**

Условные операторы

используются, когда нужно выполнить часть кода, в зависимости от какого-то условия

В зависимости от того, сколько условий, можно выделить:

- `If …` - используется, когда есть только одно условию по которому нужно выполнить задачу;
- `If … else …` - используется, когда есть также одно условие, но выполнить нужно всё что иначе;
- `If … elif … else …` - используется, когда есть много условий и иначе.

Условные операторы

❗ Действия, которые выполняются в конструкции, пишутся с табуляцией. Конструкцию разрывать нельзя.

```
if num1 > 0:
    print(f"{num1} больше 0")
elif num1 >= -10:
    print(f"{num1} в диапазоне от -10 включительно до 0")
else:
    print(f"{num1} все остальное (меньше -10)")
```

Условные операторы

❗ Такое условие будет срабатывать всегда.

```
if True:
    print('Hello!')
```

Логические операторы

- **`and`** – логический оператор (и)
- **`or`** – логический оператор (или)
- **`not`** – логический оператор (не)

```
a = 3 > 3 and 10 > 2

b = 3 < 3 or 10 > 2

c = not 1
```

Операторы принадлежности

- **`in`** – логический оператор (в)
- **`not in`** – логический оператор (нет в)

```
bool1 = "hello" in "hello world"
bool2 = "hello" not in "hello world"
```

Операторы тождественности

Такие операторы проверяют, одинаковая ли ссылка на ячейку памяти у двух переменных.

- **`is`** – логический оператор (это)
- **`is not`** – логический оператор (это не)

```
x = 2
y = 2
z = "2"
bool3 = x is y
bool4 = x is not z
```

! Для того, чтобы посмотреть в какой ячейке памяти хранится переменная, можно воспользоваться функцией `id(a)`, a - переменная, ссылку которой выведет функция.

Обработчики исключений

Исключениями (*exceptions*) в языках программирования называют проблемы, возникающие в ходе выполнения программы. Типичным примером исключения является деление на ноль, невозможность считать данные из устройства, отсутствие доступной памяти, доступ к закрытой области памяти и т.п.

Для обработки таких ситуаций, как правило, предусматривается специальный механизм, который называется **обработка исключений**.

Обработчики исключений

Обработчики исключений в Python рассматриваются ещё одной конструкцией оператора ветвления:
`try: … except: …`

Если код выполняется с ошибкой, то срабатывает блок `except` и выполняется код в нём.

Обработчики исключений

Все исключения имеют свой вид и являются **классами**. В одной конструкции можно писать несколько блоков `except` и обрабатывать несколько классов исключений.

```
try:
    # a = 4/0
b = "1" + 1
except ZeroDivisionError:
    print("На ноль делить нельзя!")
except TypeError:
    print("Разные типы складывать нельзя")
```

```
try:
    4\0
except ZeroDivisionError:
    pass
```

Обработчики исключений

Когда исключения обрабатываются в *конструкции*, можно использовать блок `else` для обработки всех остальных исключений. Т.е. действие будет срабатывать для тех исключений, которые не были прописаны в блоках `except`.

```
try:
    # a = 4/0
    b = "1" + 1
except ZeroDivisionError:
    print("На ноль делить нельзя!")
except TypeError:
    print("Разные типы складывать нельзя")
else:
    print("Этот блок срабатывает для всех ошибок, кроме ZeroDivisionError и TypeError")
```

Блок `finally`

позволяет совершать действия в конце работы конструкции **`try` … `except` …** , вне зависимости от того, была ошибка или нет, т.е. `finally` срабатывает всегда.

```
try:
    # a = 4/0
    b = "1" + 1
except ZeroDivisionError:
    print("На ноль делить нельзя!")
except TypeError:
    print("Разные типы складывать нельзя")
else:
    print("Этот блок срабатывает для всех ошибок, кроме ZeroDivisionError и TypeError")
finally:
    print("Этот код будет срабатывать всегда")
```