ЯЗЫК PYTHON

Лекция 2. Условное выполнение

Комментарии

- Иногда полезно описать смысл кода
- Для этого есть специальная конструкция - комментарии
- Начинается с символа #
- Заканчивается концом строки

Код с комментариями

```
1 name = input() # вводим имя
2
3 # печатаем
4 print("Hello, " + name)
```

Расширим представления о типах

- Какие типы мы знаем: целые, "вещественные", строки
- Есть еще парочка
- И они нам пригодятся
  - Логические
  - "Ничто", None

Логические значения

- "Истина" и "ложь"
- Литералы: True и False
- Результат операций сравнения
  - Равно-неравно
  - Больше-меньше
- И некоторых других
- Операции над логическими значениями

Равенство

- Обозначается как ==
- Не путаем с присваиванием (=)
- Целое с целым
- Вещественное с вещественным
- Строка со строкой
- Логическое с логическим

Примеры на равенство

```
1 print(123 == 321) # False
2 print(123 == '123') # False
3 print(123 == 120 + 3) # True
4 print(123 == int('123')) # True
5 print('123' == '123') # True
6 print('123 ' == '123') # False
7 print(False == 'False') # False
8 print(False == False) # True
```

Не-равно

- Специально не говорю "неравенство"
- Чтобы не путать с больше-меньше
- Обозначается как !=
- Отрицание равенства
- Если равно истинно, не-равно ложно
- И наоборот

Равно и не-равно для объектов разных типов

- Как правило, объекты разных типов не равны
- Но есть исключения
- Например
  - 2 целое
  - 2.0 - вещественное
  - 2 равно 2.0
  - 1 равно True
  - 0 равно False

Равенство и идентичность

- 2 и 2.0 равны
- Но это разные объекты
- Иногда хочется понять, ссылаются ли две переменные на один объект
- Есть специальная операция - is

Пример на идентичность и равенство

```
 1 v1 = 2
 2 v2 = 2.0
 3 v3 = 2
 4
 5 print(v1 == v2) # True
 6 print(v1 == v3) # True
 7 print(v2 == v3) # True
 8
 9 print(v1 is v2) # False
10 print(v1 is v3) # True
11 print(v2 is v3) # False
```

Неравенства

- Обозначаются как <, >, <=, >=
- Целое с целым
- Вещественное с вещественным
- Строка со строкой - в лексикографическом порядке ("по алфавиту")
- Логическое с логическим - True больше, чем False

Неравенства с объектами разных типов

- Однозначно определено
- Это будет важно чуть позже
- Когда изучим списки и сортировку
- Список элементов разных типов может быть однозначно отсортирован

Логические операции

- Логическое И: and
- Логическое ИЛИ: or
- Вычисляются "лениво" ("по Маккарти")
- Если результат понятен по первому аргументу, второй не вычисляется
- Логическое НЕ: not (унарная)

Почему "ленивость" полезна

- Если один аргумент вычисляется сильно дешевле другого
- Имеет смысл поставить его слева
- И тогда более дорогой иногда можно будет не считать

Почему "ленивость" полезна

- Хотим проверить условие, в котором есть слабое место
- Например, деление на что-то
- Что может быть нулем
- Можем использовать логическое И
- И первым аргументом поставить проверку условия

Приведение к логическому типу

- Явно с помощью функции bool
- Неявно - при использовании выражения в логическом контексте
- Подробнее - чуть позже
- Уже совсем скоро

Приведение к логическому типу

- Ненулевое целое - True, 0 - False
- Ненулевое вещественное - True, 0.0 - False
- Непустая строка - True, пустая - False

"Ничто", None

- В переменной может храниться значение
- Правильнее сказать - ссылка на объект
- А если переменная уже есть
- А значения пока еще нет

"Ничто", None

- Или мы хотели что-то посчитать или найти
- И сохранить результат в переменной
- А найти не удалось
- Нужно какое-то значение, которое опишет эту ситуацию
- Ситуацию отсутствия значения

"Ничто", None

- В Python есть такое значение
- Обозначается литералом None
- У этого значения свой отдельный тип
- Он тоже называется None

"Ничто", None

- При приведении к логическому типу превращается в False
- Проверка на None: оператор is
- Можно и через равенство
- Но is - надежнее
- Подробнее - позже

Условные предложения

- Пока мы можем хранить логические значения в переменных
- И печатать
- Не густо
- Основное использование - в условных предложениях
- Исполняем код в зависимости от значения логического выражения

Вспомним одну из первых программ

- Спрашиваем имя
- И пишем `Hello, <имя>`
- А если ввели пустую строчку ?
- Как-то некрасиво получается `Hello,`
- Хочется пропустить

Пропускаем пустую строчку

```
1 name = input()
2
3 if len(name) > 0:
4     print("Hello, " + name)
```

Составные части условного предложения

- Ключевое слово `if`
- Условие - выражение с логическим значением
- Двоеточие - маркер конца условия
- Блок
  - Набор предложений
  - С одинаковым смещением
  - "Правее", чем if

Улучшим немного

- Код корректен
- Но не идиоматичен
- Условие задает логический контекст
- Воспользуемся этим

Используем логический контекст

```
1 name = input()
2
3 if name:
4     print("Hello, " + name)
```

Хотим два варианта действий

- Чисто технически - можно повторить if с инвертированным условием
- Но это громоздко и некрасиво
- Есть специальная конструкция
- Ключевое слово else
- На том же смещении, что и if

Используем два варианта действий

```
1 name = input()
2
3 if name:
4     print("Hello, " + name)
5 else:
6     print("Hello, world")
```

Хотим много вариантов

- Хотим по номеру месяца и году определить количество дней
- Проверяем номер месяца
- И високосность года
- 4 легальных варианта
- И еще вариант некорректных данных

Хотим много вариантов

- Чисто технически - можем организовать цепочку if/else
- Первый вариант - под if
- Под else - новый if
- По которым - новый if
- И т.д.

Хотим много вариантов

- Но каждый вложенный if даст дополнительное смещение
- Последние варианты уедут сильно вправо
- Что как минимум неудобно для чтения
- И вводит в заблуждение
- Потому что все условия - однородны

Хотим много вариантов

- Есть специальное ключевое слово - elif
- После него - идет условие
- Двоеточие
- И дальше - новый блок
- И таких elif может быть много
- И все - на одном смещении

Пример с elif

```
 1 n = int(input())
 2
 3 max_days = None
 4 if n == 2:
 5     max_days = 28
 6 elif n == 4 or n == 6 or n == 9 or n == 11:
 7     max_days = 30
 8 elif n < 1 or n > 12:
 9     print("Bad data !!!")
10 else:
11     max_days = 31
12 print(max_days)
```

Условное выражение

- Мы видели условные предложения
- Бывает, что условные вычисления нужны в контексте выражения
- Чтобы вычислить один из аргументов
- Можем использовать условное предложение
- И временную переменную
- Но есть более компактный вариант

Пример if как выражения

```
1 name = input()
2 print("Hello, " + (name if name else "world"))
```

Сравнения и типы данных

- Целые числа с целыми - соответствует школьным правилам и здравому смыслу
- Вещественные числа с вещественными - в целом тоже
- Но есть специальные значения
- И тонкость со сравнением
- Целые с вещественными - целое приводится к вещественному

Специальные значения

- Бесконечность
- Минус бесконечность
- "Не число"
- Любое число в программе - объект
- Специальные значения - тоже объекты

Специальные значения

- Обычному числу соответствует литерал
- Который обозначает соответствующий объект
- У специального значения нет литерала
- Но можно создать соответствующий объект
- Как минимум, два способа

Первый способ

```
1 import math
2
3 print(math.inf)
4 print(-math.inf)
5 print(math.nan)
```

Второй способ

```
1 print(float('inf'))
2 print(float('-inf'))
3 print(-float('inf'))
4 print(float('nan'))
```

Сравнения с участием специальных значений

- Обычное число меньше бесконечности
- Обычное число больше минус бесконечности
- Минус бесконечность меньше бесконечности
- Сравнение на "больше-меньше" с "не числом" всегда дает ложь

Сравнения с участием специальных значений

- Сравнение на "не равно" с "не числом" всегда дает истину
- Сравнение на "равно" с "не числом" всегда дает ложь
- В том числе - если сравниваем с "не числом"
- То есть если в переменной v хранится "не число"
- То v == v - ложь, а v != v - истина

А зачем они нужны

- Предположим, что есть программа, которая что-то считает
- Какой-то числовой показатель
- И по итогам вычислений хочется что-то сделать в зависимости от значения показателя
- Если он меньше порогового значения

А зачем они нужны

- А теперь предположим, что мы хотим менять пороговое значение
- Ничего другого не меняя
- Можно завести переменную и хранить в ней пороговое значение
- И будет что-то вроде `if result < THRESHOLD: ...`

А зачем они нужны

- А теперь предположим, что мы хотим иногда отменять пороговое значение
- То есть выполнять действие всегда
- Можем завести вторую переменную
- Которая будет определять, включен ли данный режим проверки
- И отдельно хранить конкретное пороговое значение

Как-то так

```
 1 THREAHOLD = 1234
 2 USE_THRESHOLD = True
 3
 4 .....
 5
 6 if USE_THREASHOLD:
 7     if result < THRESHOLD:
 8         do_something()
 9 else:
10     do_something()
```

А теперь - улучшим

```
1 THREAHOLD = float('inf')
2
3 .....
4
5 if result < THRESHOLD:
6     do_something()
```