Вводная лекция

Дмитрий Яковлев

ИТМО

19 сентября 2024 г.

План лекции

Введение

Про курс

Цели курса:

- Повторить основы языка программирования Python
- Научиться писать качественный код
- Углубить свои навыки разработки
- Подготовить технический бэкграунд необходимый ML инженеру

План курса

1. Введение
2. Функции и декораторы
3. Структуры данных
4. Классы
5. Инструменты анализа данных в ML
6. Web + Базы данных
7. Менеджеры контекста и тесты
8. Итераторы и генераторы
9. Параллельное программирование
10. Ассинхронное программирование
11. Классы 2
12. Инструменты глубокого обучения
13. Распределенные вычисления

Почему Python?

- Популярный язык
- Простой синтаксис
- Кроссплатформенный
- Автоматическое управлению памятью
- Встроенные типы объектов и инструменты
- Динамическая типизация
- Интерпретируемый язык

Почему Python?

Генератор чисел Фибоначчи:

```
def fibonacci(n):
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a + b
```

Автоматическо управлению памятью

Плюсы:

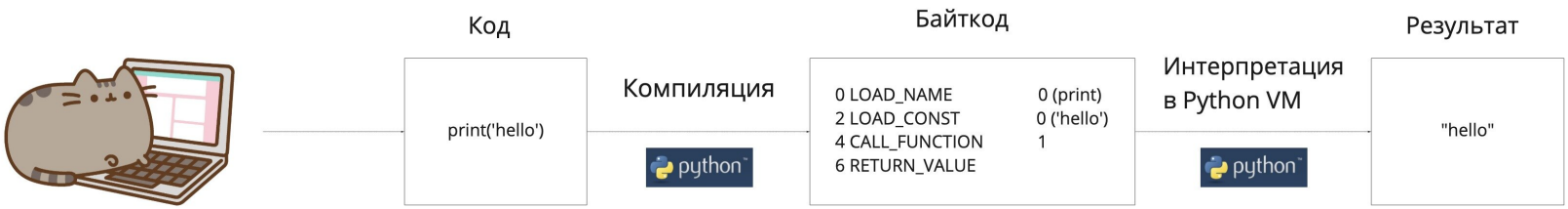
- Упрощение разработки
  - Снижение сложности кода
  - Меньше ошибок (утечки памяти, двойное освобождение)
- Повышение надежности и стабильности
  - Предотвращение утечек памяти
  - Безопасность типов (предотвращение некорректных операций с памятью)
- Кроссплатформенность и переносимость
  - Абстракция от платформы
  - Согласованное поведение на различных платформах
- Ускорение разработки и снижение стоимости поддержки
  - Быстрое прототипирование
  - Снижение затрат на отладку

Автоматическо управлению памятью

Минусы:

- Непредсказуемость производительности
  - Паузы из-за сборки мусора
  - Оверхед ресурсов
- Ограниченный контроль над памятью
  - Недостаточная гибкость
  - Зависимость от реализации на платформе
- Неоптимальное время жизни объектов
  - Задержка в освобождении памяти

Интерпретация



Пример

Писать на Python — будто писать псевдокод

$$e^x = \sum_{k=0}^{\infty} \frac{1}{k!} x^k$$

Пример

Писать на Python — будто писать псевдокод

$$e^x = \sum_{k=0}^{\infty} \frac{1}{k!} x^k$$

```
def e(x):
    sum_, k, term = 1, 0, 1
    while True:
        yield sum_
        k += 1
        term *= x / k
        sum_ += term
```

Мультипарадигмальный ЯП

- Объектно-ориентированный
- Рефлективный
- Императивный
- Функциональный

```
>>> list(map(lambda x: x * x, [i for i in range(0, 5)]))
[0, 1, 4, 9, 16]
>>> add = lambda x, y: x + y
>>> mul = lambda x, y: x * y
>>> add.__code__ = mul.__code__
>>> add(2,3)
6
```

Особенности языка

- Всё есть объект

```
>>> type(fibonacci)
<class "function">
>>> type(type(fibonacci))
<class "type">
```

- Все объекты делятся на ссылочные и атомарные
- Вместо фигурных скобок отступы

```
>>> if True:
... print("Hello world!")
  File "<stdin>", line 2
    print("Hello world!")
        ^
IndentationError: expected an indented block
```

Python 2.x от Python 3.x

- Python 2.x - прошлое (конец поддержки в 2020)
- Python 3.x - будущее (развивается)
- Основное отличие - кодирования строк с байт на Unicode

Среды разработки

- IDLE
- PyCharm
- Anaconda (IDE Spyder)
- Sublime Text, Atom, ...

Основные типы

Числовой тип

```
>>> 123 + 321
444
>>> 11.5 - 1
10.5
>>> (3+4j) * (5 + 1j)
(11+23j)
>>> 2 ** 100
1267650600228229401496703205376L
>>> 7 / 3
2.3333333333333335
>>> 7 // 3
2
>>> 7 % 3
1
```

Строки

```
>>> "Hello world!"
'Hello world!''
>>> 'Hello world!'
'Hello world!''
>>> dna = "AGCT"
>>> dna[1]
'G'
>>> len(dna)
4
>>> "A" + "GC" + "T"
'AGCT'
>>> dna[0] = "A"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: "str" object does not support item assignment
```

Список

```
>>> []
[]
>>> list()
[]
>>> dna = ["A", "G", "C", "T"]
>>> len(dna)
4
>>> dna
["A", "G", "C", "T"]
>>> dna[0] = "T"
>>> dna
["T", "G", "C", "T"]
>>> dna.append("A")
>>> del dna[0]
>>> dna
["G", "C", "T", "A"]
```

Кортеж (1)

```
>>> tuple()
()
>>> T = (3, [25, 54, 123], "GGCGAT")
>>> T[0]
3
>>> T[0] = 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: "tuple" object does not support item assignment
>>> del T[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: "tuple" object doesn"t support item deletion
>>> (3, [25, 54, 123], "GGCGAT") + (23, 25)
(3, [25, 54, 123], "GGCGAT", 23, 25)
```

Кортеж (2)

- Пустой кортеж - синглтон

```
>>> a = ()
>>> b = ()
>>> a is b
True
>>> id(a)
1731302785096
>>> id(b)
1731302785096
>>> a = (1, 2)
>>> id(a)
1731313457288
```

Сложение кортежей (1)

Модуль `dis` позволяет декомпилировать байт-код Python.

```
>>> import dis
>>> def f():
...     a = (1, 2) + (2, 3)
...     return a
...
>>> dis.dis(f)
  2           0 LOAD_CONST               1 ((1, 2, 2, 3))
              2 STORE_FAST               0 (a)

  3           4 LOAD_FAST                0 (a)
              6 RETURN_VALUE
```

Сложение кортежей (2)

```
>>> import dis
>>> def f():
...     b = (1, 2)
...     a = b + (2, 3)
...     return a
>>> dis.dis(f)
  2           0 LOAD_CONST               1 ((1, 2))
              2 STORE_FAST               0 (b)

  3           4 LOAD_FAST                0 (b)
              6 LOAD_CONST               2 ((2, 3))
              8 BINARY_ADD
             10 STORE_FAST               1 (a)

  4          12 LOAD_FAST                1 (a)
             14 RETURN_VALUE
```

Сложение строк (1)

```
>>> import dis
>>>
>>> def f():
...     a = "HelloWorld" is "Hello" + "World"
...
>>> dis.dis(f)
  2           0 LOAD_CONST               1 ('HelloWorld')
              2 LOAD_CONST               1 ('HelloWorld')
              4 COMPARE_OP               8 (is)
              6 STORE_FAST               0 (a)
              8 LOAD_CONST               0 (None)
             10 RETURN_VALUE
```

Сложение строк (2)

```
>>> import random as rd
>>>
>>> N = 10
>>> random_list = [rd.choice(["a", "b"]) for _ in range(N)]
>>> line = "".join(random_list)
>>> concatenated_line = line[:(N // 2)] + line[(N // 2):]
>>> print(line, id(line))
bbabaaaaaa 1731313887472
>>> print(concatenated_line, id(concatenated_line))
bbabaaaaaa 1731313937584
```

Множество

```
>>> nucleotids = set("AGCT")
>>> nucleotids
{"G", "C", "A", "T"}
>>> nucleotids.discard("T")
>>> nucleotids
{"G", "C", "A"}
>>> nucleotids.add("T")
>>> nucleotids
{"G", "C", "A", "T"}
>>> {1, 2, 3} | {2, 3, 4}
{1, 2, 3, 4}
>>> {1, 2, 3} & {2, 3, 4}
{2, 3}
>>> {1, 2, 3} - {2, 3, 4}
{1}
```

Словарь

```
>>> {}
{}
>>> dict()
{}
>>> genes = {"BCL-1" : 145, "BCL-2" : 134, "BCL-3" : 155}
>>> genes
{"BCL-1": 145, "BCL-3": 155, "BCL-2": 134}
>>> genes["BCL-4"] = 188
>>> genes["BCL-3"] = 120
>>> del genes["BCL-2"]
>>> genes
{"BCL-1": 145, "BCL-3": 120, "BCL-4": 188}
>>> genes.keys()
dict_keys(["BCL-1", "BCL-3", "BCL-4"])
>>> genes.values()
dict_values([145, 120, 188])
```

Логический тип

```
>>> True
True
>>> False
False
>>> True + 11
12
>>> False + 1
1
>>> True == False
False
```

Тип None

```
>>> None
>>> None is None
True
>>> None == None
True
>>> None != None
False
```

Управляющие конструкции

Управляющая конструкция: if (1)

```
>>> dna = "AGTAGTGCTATAAA"
>>> if "TATAA" in dna:
...     print("TATA box")
... elif len(dna) > 5:
...     print("correct dna")
... else:
...     pass
...
TATA box
>>> if True:
... print("Good!")
  File "<stdin>", line 2
    print("Good!")
        ^
IndentationError: expected an indented block
```

Управляющая конструкция: if (2)

```
>>> if 10 < len(dna) < 20:
...     print("Correct!")
...
Correct!

>>> if dna[0] == "A" or dna[5] == "G":
...     print("Correct!")
... elif dna[1] == "G" and dna[2] == "C":
...     dna[1] = "T"
... else:
...     pass
...
Correct!
```

Управляющая конструкция: for

```
>>> for x in ["A", "G", "C", "T"]:
...     print(x, end = " ")
...
A G C T

>>> genes = {"BCL-1" : 145, "BCL-2" : 134, "BCL-3" : 155}
>>> for gene in genes:
...     print(gene, end=" ")
...
BCL-1 BCL-3 BCL-2

>>> total = 0
>>> for x in range(5):
...     total = total + x
...
>>> total
10
```

Управляющая конструкция: while

```
>>> while x:
...     x = x - 1
...     if x % 2 == 0:
...         print(x, end=" ")
...
8 6 4 2 0

>>> while False:
...     print("Never")
... else:
...     print("Ever")
...
Ever
```

Если не было вызова `break`, то всегда выполняется `else`

Операторы: in и not in

```
>>> "A" in "AGCT"
True
>>> "A" in ["A", "G", "C", "T"]
True
>>> 5 not in ["A", "G", "C", "T"]
True
```

The Zen of Python

```
>>> import this
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren"t special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you"re Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it"s a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let"s do more of those!
```

Дзен Питона

```
>>> import this
Красивое лучше, чем уродливое.
Явное лучше, чем неявное.
Простое лучше, чем сложное.
Сложное лучше, чем запутанное.
Плоское лучше, чем вложенное.
Разреженное лучше, чем плотное.
Читаемость имеет значение.
Особые случаи не настолько особые, чтобы нарушать правила.
При этом практичность важнее безупречности.
Ошибки никогда не должны замалчиваться.
Если они не замалчиваются явно.
Встретив двусмысленность, отбрось искушение угадать.
Должен существовать один и, желательно, только один очевидный способ сд
Хотя он поначалу может быть и не очевиден, если вы не голландец1.
Сейчас лучше, чем никогда.
Хотя никогда зачастую лучше, чем прямо сейчас.
Если реализацию сложно объяснить - идея плоха.
Если реализацию легко объяснить - идея, возможно, хороша.
Пространства имён - отличная штука! Будем делать их больше!
```

Резюме

- Используем Python версии 3.12+
- Только отступы
- Атомарные типы - bool, int, float, complex
- Ссылочные типы - все остальные
- Удобные операторы in и not in

Спасибо за внимание!

Ресурсы

Картинка