

5

МОДЕЛЬ ДАННЫХ В PYTHON

5.1 Объекты и их характеристики

5.2 Управление памятью

5.3 Особенности типизации

5.4 Иерархия встроенных типов данных

5.5 Классификация операций над встроенными типами данных

5.6 Терминология

5.7 Контрольные вопросы и упражнения



5

Модель данных в Python

5.1 Объекты и их характеристики

В Python-программе любые данные представляются в виде объектов (англ. object). Каждый объект характеризуется тройкой: id, тип и значение.

Id (англ. identity) объекта – указатель (англ. pointer) на область памяти, где находится объект (для реализации CPython). Id объекта гарантировано будет уникальным в течение его жизни. В Python для работы с id объекта используется функция `id(x)`, которая возвращает адрес объекта `x` в памяти, и оператор `is`, который проверяет на равенство id двух объектов. Например:

```
>>> a, b, c = 'Python', 'Python', 10
>>> id(a)
26140160
>>> id(b)
26140160
>>> id(c)
1362519280
>>> a is b
True
>>> a is c
False
>>> b is c
False
```

Из примера видно, что объекты с именами `a` и `b` имеют одинаковый id, так как ссылаются на один и тот же объект – объект со значением 'Python'.

Тип (англ. type) объекта – характеристика объекта, которая определяет его внутреннее представление, а также операции, которые им поддерживаются и множество допустимых значений. Для создания объектов могут быть использованы как встроенные типы данных, так и пользовательские. Объект определенного типа называют экземпляром или инстансом этого типа. В Python для работы с типом объекта используются встроенные функции `type(x)` и `isinstance(x, classinfo)`. Функция `type(x)` возвращает тип объекта `x`, а функция `isinstance(x, classinfo)` проверяет принадлежность объекта к одному из типов `classinfo`. Например:

```
>>> a = True
>>> type(a)
<class 'bool'>
>>> b = 1
```

```
>>> type(b)
<class 'int'>
>>> isinstance(a, int)
True
>>> isinstance(a, str)
False
>>> isinstance(a, (bool, str))
True
```

Из примера видно, что тип данных `bool` является производным типом от `int`.

Значение (англ. *value*) объекта – непосредственные данные, хранящиеся в объекте.

В течение жизни объекта его `id` и тип постоянны, а значение некоторых объектов может изменяться. Если значение объекта может изменяться, то такой объект называется мутируемым (англ. *mutable*). Если значение объекта не может изменяться, то такой объект называется немутуруемым (англ. *immutable*). Изменчивость значения объекта определяется его типом, например объекты встроенного типа `str` являются не мутируемыми, а `list` – мутируемыми.

5.2 Управление памятью

В Python выделение и освобождение памяти под объекты осуществляется автоматически. Для корректного управления памятью, в реализации CPython используется схема подсчета ссылок (англ. *reference-counting scheme*). С помощью данной схемы интерпретатор следит за количеством активных ссылок (англ. *reference*) на каждый объект программы в текущий момент времени. Значение счетчика ссылок на объект увеличивается при создании новой ссылки, и уменьшается, если ссылка на объект по какой-либо причине пропадает. Новая ссылка на объект создается, например, в следующих случаях:

- связывание переменной с объектом с помощью оператора присваивания;
- добавление объекта в некоторую последовательность;
- передача объекта в функцию.

Пример создания нескольких ссылок на объект:

```
>>> # Создание объекта со значением 'Program' и одной ссылкой
>>> a = 'Program'
>>> # Увеличиваем счетчик ссылок на 1
>>> b = 'Program'
>>> # Увеличиваем счетчик ссылок на 1
>>> c = a
>>> # Увеличиваем счетчик ссылок на 3
>>> d = [a, b, c]
```

В примере создано только два объекта: объект строкового типа со значением `'Program'` и объект списочного типа с тремя элементами. Важно понимать, что в программе объект со значением `'Program'` был создан только один раз и с ним в итоге связано 6 ссылок:

переменные `a`, `b`, `c`, и элементы списка `d[0]`, `d[1]`, `d[2]`. Убедиться в том, что переменные `a`, `b`, `c` и три элемента списка `d` ссылаются на один и тот же объект можно с помощью следующего неравенства:

```
>>> id(a) == id(b) == id(c) == id(d[0]) == id(d[1]) == id(d[2])
True
```

Примеры удаления ссылки на объект:

- связывание переменной с другим объектом;
- явное уничтожение переменной с помощью оператора `del`;
- поток выполнения покидает блок кода, в котором был определен объект.

Проверить количество активных ссылок на объект в текущий момент времени можно с помощью функции `getrefcount(x)`, которая находится в модуле `sys`. Возвращаемое функцией значение обычно на единицу больше от ожидаемого, так как создается новая ссылка (временная) на объект при его передаче в функцию. Пример:

```
>>> a = 'Program'
>>> b = 'Program'
>>> c = a
>>> d = [a, b, c]
>>> # Подключаем модуль sys
>>> import sys
>>> # При вызове функции getrefcount(x) \
>>> # создается временная ссылка на объект x.
>>> sys.getrefcount(a)
7
>>> # Удаляем список d и \
>>> # вместе с ним три ссылки на объект 'Program'.
>>> del d
>>> sys.getrefcount(a)
4
>>> del c
>>> sys.getrefcount(a)
3
>>> del b
>>> sys.getrefcount(a)
2
>>> del a
```

В примере с помощью оператора `del` последовательно удаляются переменные `d`, `c`, `b` и `a`, а вместе с ними и ссылки на объект со значением `'Program'`. По значениям, которые возвращает функция `getrefcount(x)` видно, как значение счетчика ссылок на объект после каждого вызова оператора `del` уменьшается. После выполнения последнего оператора `del` с переменной `a` значение счетчика будет установлено в ноль и память, которую занимал объект со значением `'Program'`, будет освобождена. Такой процесс освобождения памяти называется сборкой мусора (англ. *garbage collection*).

При этом следует отметить, что интерпретатор Python стремится минимизировать количество потребляемой памяти и для некоторых значений неизменяемых типов данных заранее создает объекты. Например:

```
>>> import sys
>>> sys.getrefcount(1)
690
>>> sys.getrefcount(2)
370
>>> sys.getrefcount('Python')
3
```

5.3 Особенности типизации

Как было сказано выше, каждый объект характеризуется типом данных. Операция назначения объекту типа данных называется типизацией (англ. type checking). Типизированные языки программирования принято разделять на языки со статической / динамической и сильной / слабой типизацией. При статической типизации (англ. static typing) тип данных объекта устанавливается на этапе компиляции программы, а при динамической (англ. dynamic typing) – на этапе выполнения. Сильная типизация (англ. strong typing), в отличие от слабой (англ. weak typing), предполагает, что во время выполнения программы должно выполняться согласование типов объектов.

Python относится к языкам программирования с сильной динамической типизацией.

Учитывая особенности типизации и управления памятью в Python, можно сделать следующие выводы:

- объект «знает» свой id, к какому типу он относится и свое значение;
- определение переменной невозможно без ее связывания с конкретным объектом с помощью оператора присваивания;
- в текущий момент времени переменная может ссылаться только на один объект;
- так как тип данных является характеристикой объекта, поэтому в разные моменты времени переменная может ссылаться на разные объекты разных типов данных, например:

```
>>> # Переменная x ссылается на объект целого типа \
>>> # со значением 100.
>>> x = 100
>>> type(x), id(x)
(<class 'int'>, 1373334160)

>>> # Переменная x ссылается на объект вещественного типа \
>>> # со значением 100.0.
>>> x = 100.0
>>> type(x), id(x)
(<class 'float'>, 52686832)

>>> # Переменная x ссылается на объект строкового типа \
>>> # со значением '100'.
>>> x = '100'
```

```
>>> type(x), id(x)
(<class 'str'>, 56228384)
```

- в выражениях можно использовать только те объекты, для которых установлены правила согласования, например:

```
>>> x = 10
>>> y = 45.8
>>> # Для объектов целого и вещественного типов \
>>> # установлены правила согласованности.
>>> z = x + y
>>> z, type(z)
(55.8, <class 'float'>)
```

```
>>> x = 10
>>> y = '45.8'
>>> # Для объектов целого и строкового типов \
>>> # не определены правила согласованности.
>>> z = x + y
```

TypeError: unsupported operand type(s) for +: 'int' and 'str'

Как видно из примера, использование объектов в одном выражении, для которых не установлены правила согласованности в Python запрещено. В таких случаях, если возможно, необходимо использовать явное приведение типов, например

```
>>> x = 10
>>> y = '45.8'
>>> z = x + float(y)
>>> z, type(z)
(55.8, <class 'float'>)
```

5.4 Иерархия встроенных типов данных

В Python существует большое количество стандартных (встроенных) типов данных, которые характеризуются высокой эффективностью и производительностью. Все встроенные типы данных сгруппированы в несколько категорий: числовые типы, последовательности, множества, отображения и специальные. Описание всех встроенных типов (название, класс, примеры) из категорий числовые типы, последовательности, множества и отображения приведены в таблице 5.1. К категории специальные типы обычно относят функции, классы, файлы и модули. Некоторые встроенные типы относятся к классу мутируемых, а некоторые – к немутуруемым. Во встроенном пространстве имен Python также существует несколько стандартных объектов-констант, например:

- False – значение логического типа, которое соответствует значению «ложь»;
- True – значение логического типа, которое соответствует значению «правда»;
- None – единственный представитель типа NoneType, который означает «ничего», «объект без значения».

Таблица 5.1 – Встроенные типы данных в Python

Название	Класс	Мутируемый / немутуируемый	Пример
<i>Числовые типы (англ. numbers)</i>			
Целое число	int	немутуируемый	<pre>>>> a = int(10) >>> b = 10 >>> a, b (10, 10)</pre>
Логическое значений	bool	немутуируемый	<pre>>>> a = True >>> b = False >>> a, b (True, False)</pre>
Число с плавающей точкой	float	немутуируемый	<pre>>>> a = float(10.5) >>> b = 10.5 >>> a, b (10.5, 10.5)</pre>
Комплексное число	complex	немутуируемый	<pre>>>> a = complex(2, 3) >>> b = 2 + 3j >>> a, b ((2+3j), (2+3j))</pre>
<i>Последовательности (англ. sequences)</i>			
Строка	str	немутуируемый	<pre>>>> a = str('python') >>> b = 'Python' >>> a, b ('python', 'Python')</pre>
Кортеж	tuple	немутуируемый	<pre>>>> a = tuple([10, '10']) >>> b = (10, '10') >>> a, b ((10, '10'), (10, '10'))</pre>
Последовательность байт	bytes	немутуируемый	<pre>>>> a = bytes(b'Python') >>> b = b'Python' >>> a, b (b'Python', b'Python')</pre>
Список	list	мутуируемый	<pre>>>> a = list([10, '10']) >>> b = [10, '10'] >>> a, b ([10, '10'], [10, '10'])</pre>
Последовательность байт	bytearray	мутуируемый	<pre>>>> a = bytearray(b'Python') >>> a bytearray(b'Python')</pre>
Перечисление	range	немутуируемый	<pre>>>> a = range(1, 10, 2) >>> a range(1, 10, 2)</pre>
<i>Множества (англ. set types)</i>			
Множество	set	мутуируемый	<pre>>>> a = set([1, 2, 2, 3]) >>> b = {1, 2, 2, 3} >>> a, b ({1, 2, 3}, {1, 2, 3})</pre>
Замороженное множество	frozenset	немутуируемый	<pre>>>> a = frozenset([1, 2, 2, 3]) >>> a frozenset({1, 2, 3})</pre>
<i>Отображения (англ. mappings)</i>			
Словарь	dict	мутуируемый	<pre>>>> a = dict(one=1, two=2) >>> b = {'one':1, 'two':2} >>> a, b ({'one': 1, 'two': 2}, {'one': 1, 'two': 2})</pre>

5.5 Классификация операций над встроенными типами данных

Операция (англ. operation) – некоторое действие над данными. В программировании выполняемое действие принято называть оператором (англ. operator), а данные – операндами (англ. operand). Операции обычно классифицируются по количеству операндов, форме записи или типу оператора.

По количеству операндов операции делятся на:

- унарные (англ. unary) — в операции участвует только один операнд;
- бинарные (англ. binary) — в операции участвует два операнда;
- тернарные (англ. ternary) — в операции участвует три операнда.

Пример операций с различным количеством операндов:

```
>>> # Унарная операция.
>>> ~16
-17
>>> # Бинарная операция.
>>> 35 + 25
60
>>> # Тернарная операция.
>>> 7 ** 3 if 10 ** 3 > 2 ** 10 else 7 ** (-3)
0.0029154518950437317
```

По форме записи существует 3 варианта синтаксиса операций:

- префиксная (польская) запись (англ. prefix) – форма записи, при которой оператор располагается перед операндами;
- инфиксная запись (англ. infix) – форма записи, при которой операторы располагаются между операндами, на которые они воздействуют;
- постфиксная (обратная польская) запись (англ. postfix) – форма записи, при которой операнды располагаются перед операторами.

```
>>> # Префиксная запись операции.
>>> ~16
-17
>>> # Инфиксная запись операции.
>>> 35 + 25
60
>>> # Постфиксная запись операции.
>>> 'Hello, world!'[1:-1:2]
'el, wrd'
```

По типу оператора обычно выделяют следующие категории операций:

- операции присваивания (англ. assignment) (операторы: =, +=, -=, *=, /=, %=, **=, //=);
- арифметические операции (англ. arithmetic) (операторы: +, -, *, /, %, **, //);
- операции сравнения (англ. relational) (операторы: ==, !=, >, <, >=, <=, in, not in, is, is not);

- логические операции (англ. logical) (операторы: and, or, not);
- побитовые операции (англ. bitwise) (операторы: &, |, ^, ~, <<, >>);

В Python всем операторам соответствует определенная функция из модуля `operator`, что позволяет заменить операцию на вызов функции, например бинарному оператору «+» соответствует функция `add(a, b)`:

```
>>> a = 10
>>> b = 10
>>> a + b
20
>>> import operator
>>> operator.add(a, b)
20
```

Такое решение позволяет в некоторых случаях, используя функциональный подход, значительно сократить количество кода, например:

```
>>> a = [1, 2, 3, 4, 5]
>>> # Итерационное решение.
>>> res = 1
>>> for i in a:
>>>     res *= i
>>> res
120

>>> # Функциональное решение.
>>> from functools import reduce
>>> from operator import mul
>>> reduce(mul, a)
120
```

Как видно из примера, решение задачи вычисления произведения всех элементов списка во втором варианте читабельнее, чем в первом.

5.6

Терминология

объект	множество
указатель	отображение
переменная	типизация
ссылка	сборка мусора
тип данных	операция
встроенный тип данных	оператор
число	операнд
последовательность	

5.7 Контрольные вопросы и упражнения

Контрольные вопросы:

1. Дайте определение понятию «объект». Перечислите основные характеристики объектов в Python.
2. Дайте определение понятию «переменная». Перечислите основные свойства переменных в Python.
3. Приведите классификацию встроенных типов данных в Python.
4. Какие встроенные функции в Python позволяют работать с типом и id объекта?
5. Опишите особенности управления памятью в Python.
6. Как в Python можно определить количество ссылок на объект?
7. Опишите основные особенности типизации в Python.
8. Дайте определение понятиям «операция», «оператор» и «операнд». Перечислите основные свойства переменных в Python.
9. Приведите классификацию операций над встроенными типами данных.
10. Опишите назначение модуля operator.

Вопросы для самостоятельной работы:

1. Механизмы проверки типа объекта.
2. Функции модуля operator.
3. Циклические ссылки.
4. Слабые ссылки в Python
5. Интерфейс управления сборщиком мусора (модуль gc).