КМБ: шпаргалка Python 3

ОГЛАВЛЕНИЕ

Глава 1. Введение 3

Глава 2. Примитивы 4

Числа 4

Строки 6

Логические значения 7

Глава 3. Коллекции 10

Списки 10

Словари 11

Глава 4. Операторы управления вычислениями 12

Оператор IF 12

Циклы 13

Глава 5. Функции 15

ГЛАВА 1. ВВЕДЕНИЕ

Python – прекрасный язык. Он прост и весел в освоении, а его синтаксис прост и элегантен. Python является популярным выбором для начинающих, но все же достаточно мощным, чтобы поддержать некоторые из самых популярных в мире продуктов и приложений от таких компаний, как NASA, Google, Mozilla, Cisco, Microsoft и Instagram, среди прочих. Какой бы ни была цель, дизайн Python делает программирование почти таким же естественным, как письмо на английском языке

ГЛАВА 2. ПРИМИТИВЫ

ЧИСЛА

В Python определены целые числа и числа с плавающей точкой. Целые числа — это просто целые числа, такие как 314, 500 и 716. Числа с плавающей точкой – это дробные числами, такие как 3,14, 2,867, 76,88887. Метод type можно использовать для проверки типа объекта.

```
>>> type(3)
<class 'int'>
>>> type(3.14)
<class 'float'>
>>> pi=3.14
>>> type(pi)
<class 'float'>
```

В последнем примере pi — это имя переменной, а 3,14 — значение.

Можно использовать основные математические операторы:

```
>>>3 + 3
6
>>>3 - 3
0
>>>3 / 3
1.0
>>>3 / 2
1.5
>>>3 * 3
9
>>>3 ** 3
27
>>>num = 3
>>>num = num - 1
>>>print(num)
```

```
2
>>>num = num + 10
>>>print(num)
12
>>>num += 10
>>>print(num)
22
>>>num -= 12
>>>print(num)
10
>>>num *=10
>>>num
100
```

Существует также специальный оператор под названием деление по модулю, %, который возвращает остаток после целочисленного деление.

```
>>>10 % 3
1
```

Одним из распространенных применений деления по модулю является определение того, делится ли число нацело на другое число. Например, мы знаем, что число чётное, если остаток от деления на 2 равен 0.

```
>>>10 % 2
0
>>>12 % 2
0
```

Наконец, обязательно используйте скобки для обеспечения нужного приоритета выполнения операций.

```
>>>(2+3) * 5
25
>>>2 + 3 * 5
17
```

СТРОКИ

Строки довольно часто используются в Python. Строки, это просто последовательность символов – это все, что вы можете набрать на клавиатуре одним нажатием клавиши, например, буква, цифра или косая черта.

Одинарные и двойные кавычки в Python одинаковы, и обозначают начало и конец строки.

```
>>>"список строк"
' список строк '
>>>' список строк '
' список строк '
```

Что делать, если в середине строки есть кавычка? Python нуждается в помощи, чтобы распознавать кавычки как часть языка, а не как часть языка Python.

```
>>>"Я не могу этого сделать"
' Я не могу этого сделать '
>>>" Он сказал мне \"нет\" "
' Он сказал мне "нет"'
```

Теперь вы также можете объединять (объединять) строки с использованием переменных.

```
>>>a = "первый"
>>>b = "второй"
>>>a + b
'первыйвторой'
```

Если вы хотите пробел между ними, вы можете изменить первое слово на слово с пробелом после.

```
>>>a = "первый "
>>>a + b
```

```
3 'первый второй'
```

Существуют на выбор различные строковые методы, такие как upper(), lower(), replace() и count().

upper() делает именно то, что это означает – меняет все буквы в строке на прописные.

```
1 >>>str = 'woah!'
2 >>>str.upper()
3 'WOAH!'
```

Сможете ли вы догадаться, что делает **lower()?**

```
1 >>>str = 'WOAH!'
2 >>>str.lower()
3 'woah!'
```

replace() позволяет заменить любой символ другим символом.

```
1 >>>str = 'rule'
2 >>>str.replace('r', 'm')
3 'mule'
```

Наконец, count() позволяет узнать, сколько раз определенный символ появляется в строке..

```
1 >>>number_list = ['one', 'two', 'one', 'two', 'two']
2 >>>number_list.count('two')
3 3
```

С помощью метода format()также можно форматировать/создавать строки.

```
1 >>>"{0} is a lot of {1}".format("Python", "fun!")
2 'Python is a lot of fun!'
```

ЛОГИЧЕСКИЕ ЗНАЧЕНИЯ

Логические значения – это просто True или False.

Проверьте, равно ли значение другому значению с двумя знаками равенства **==**.

```
>>>10 == 10
True
>>>10 == 11
False
>>>"jack" == "jack"
True
>>>"jack" == "jake"
False
```

Для проверки неравенства используйте **!=**.

```
>>>10 != 10
False
>>>10 != 11
True
>>>"jack" != "jack"
False
>>>"jack" != "jake"
True
```

Вы также можете протестировать >, <, >= и **<=**.

```
>>>10 > 10
False
>>>10 < 11
True
>>>10 >= 10
True
>>>10 <= 11
True
>>>10 <= 10<0
False
>>>10 <= 10 < 11
```

```
True
>>>"jack" > "jack"
False
>>>"jack" >= "jack"
True
```

ГЛАВА 3. КОЛЛЕКЦИИ

СПИСКИ

Списки — это контейнеры для хранения значений.

```
>>>fruits = ['яблоко', 'лимон', 'апельсин', 'виноград']
>>>fruits
['яблоко', 'лимон', 'апельсин', 'виноград']
```

Для доступа к элементам списка можно использовать связанные с ними индексы. Просто помните, что список начинается с 0, а не с 1.

```
>>>fruits[2]‘’
апельсин
```

Если список длинный, и нужно считать с конца, то это можно легко сделать.

```
>>>fruits[-2]‘’
апельсин
```

Иногда списки могут быть длинными и необходимо знать, сколько элементов в списке – используйте функцию len().

```
>>>len(fruits)
4
```

Используйте append() для добавления нового элемента в конец списка и pop() для удаления элемента с конца.

```
>>>fruits.append('черника')
>>>fruits
['яблоко', 'лимон', 'апельсин', 'виноград', 'черника']
>>>fruits.append('помидор')
>>>fruits
['яблоко', 'лимон', 'апельсин', 'виноград', 'черника', 'помидор']
>>>fruits.pop()
'помидор'
>>>fruits
```

```
10  ['яблоко', 'лимон', 'апельсин', 'виноград', 'черника']
```

Проверьте, есть ли значение в списке.

```
1  >>>'яблоко' in fruits
2  True
3  >>>'помидор' in fruits
4  False
```

СЛОВАРИ

Словарь оптимизирует поиск элементов. Он использует пары ключ-значение, а не числа в качестве идентификатора элемента. Каждый ключ должен иметь значение и его можно использовать для поиска значения.

```
1  >>>words = {'яблоко': 'красное', 'лимон': 'желтый'}
2  >>>words
3  {'яблоко': 'красное', 'лимон': 'желтый'}
4  >>>words['яблоко']
5  'красное'
6  >>>words['лимон']
7  'желтый'
```

Также можно работать с числами..

```
1  >>>dict = {'one': 1, 'two': 2}
2  >>>dict
3  {'one': 1, 'two': 2}
```

Вывод всех ключей с помощью keys() и всех значений с values().

```
1  >>>words.keys()
2  dict_keys(['яблоко', 'лимон'])
3  >>>words.values()
4  dict_values(['красное', 'желтый'])
```

ГЛАВА 4. ОПЕРАТОРЫ УПРАВЛЕНИЯ ВЫЧИСЛЕНИЯМИ

ОПЕРАТОР IF

Оператор **if** используется для проверки истинности условия. По сути, если условие истинно, интерпретатор Python запускает блок операторов, называемый **if-block**. Если оператор **false**, интерпретатор пропускает блок **if** и обрабатывает другой блок операторов, называемый **else-block**. Предложение **else** не обязательно.

Пара небольших примеров.

```
>>>num = 20
>>>if num == 20:
...         print('the number is 20')
...    else:

...         print('the number is not 20')
...
The number is 20
>>>num = 21
>>>if num == 20:
...         print('the number is 20')
...  else:
... print('the number is not 20')
...
The number is not 20
```

Можно также добавить предложение **elif**, чтобы добавить еще одно условие для проверки.

```
>>>num = 21
>>>if num == 20:
...         print('число 20')
...    elif num > 20:
...         print(«Число больше 20»)
```

```
...    else:
...        print(«число меньше 20»)
...
Число больше 20
```

ЦИКЛЫ

В Python используется 2 вида циклов – цикл for и цикл while. Цикл for используются, когда у вас есть фрагмент кода, который вы хотите повторить n раз. Он также обычно используются для циклов или итерации по спискам.

```
>>>colors = ['red', 'green', 'blue']
>>>colors
['red', 'green', 'blue']
>>>for color in colors:
...        print('I love ' + color)
...
I love red
I love green
I love blue
```

Цикл for, используются для повторяющихся разделов кода. В отличие от цикла for, цикл while продолжается до тех пор, пока не будет выполнено определенное условие.

```
>>> num = 1
>>> num
1
>>> while num <= 5:
...        print(num)
...        num += 1
...
1
2
```

```
3
4
5
```

ГЛАВА 5. ФУНКЦИИ

Функции — это блоки повторного использования кода, выполняющие одну задачу.

Def используется для определения (или создания) новой функции, а затем вызываете функцию, добавляя параметры к имени функции.

```
1  >>def multiply(num1, num2):
2  ...        return num1 * num2
3  ...
4  >>>multiply(2, 2)
5  4
```

Можно также задать значения параметров по умолчанию.

```
1  >>>def multiply(num1, num2=10):
2  ...        return num1 * num2
3  ...
4  >>>multiply(2)
5  20
```

Гораздо больше полезного на сайте PYTHON курс молодого бойца по адресу https://chel-center.ru/python-yfc/