# Операторы

# Концептуальная иерархия Python

1. Программы состоят из модулей.
2. Модули содержат операторы.
3. Операторы содержат выражения.
4. Выражения создают и обрабатывают объекты.

# Операторы Python

| Оператор | Роль | Пример |
|---|---|---|
| Присваивания | Создание ссылок | `a, b = 'good', 'bad'` |
| Вызовы и другие выражения | Выполнение функций | `log.write("spam, ham")` |
| Вызовы `print` | Вывод объектов | `print('The Killer', joke)` |
| `if/elif/else` | Выбор действий | `if "python" in text:`<br>`  print(text)` |
| `for/else` | Итерация | `for x in mylist:`<br>`  print(x)` |
| `while/else` | Универсальные циклы | `while X > Y:`<br>`  print('hello')` |
| `pass` | Пустой заполнитель | `while True:`<br>`  pass` |
| `break` | Выход из цикла | `while True:`<br>`  if exittest(): break` |
| `continue` | Продолжение цикла | `while True:`<br>`  if skiptest(): continue` |
| `def` | Функции и методы | `def f(a, b, c=1, *d):`<br>`  print(a+b+c+d[0])` |
| `return` | Результаты функций | `def f(a, b, c=1, *d):`<br>`  return a+b+c+d[0]` |
| `yield` | Генераторные функции | `def gen(n):`<br>`  for i in n: yield i*2` |

# Операторы Python

| Оператор | Роль | Пример |
|---|---|---|
| `global` | Пространства имен | `x = 'old'`<br>`def function():`<br>`  global x, y; x = 'new'` |
| `nonlocal` | Пространства имен (Python 3.X) | `def outer():`<br>`  x = 'old'`<br>`  def function():`<br>`  nonlocal x; x = 'new'` |
| `import` | Доступ к модулям | `import sys` |
| `from` | Доступ к атрибутам | `from sys import stdin` |
| `class` | Построение объектов | `class Subclass(Superclass):`<br>`  staticData = []`<br>`  def method(self): pass` |
| `try/except/ finally` | Перехват исключений | `try:`<br>`  action()`<br>`except:`<br>`  print('action error')` |
| `raise` | Генерация исключений | `raise EndSearch(location)` |
| `assert` | Отладочные проверки | `assert X > Y, 'X too small'` |
| `with/as` | Диспетчеры контекста (Python 3.X, 2.6+) | `with open('data') as myfile:`<br>`  process(myfile)` |
| `del` | Удаление ссылок | `del data[k]`<br>`del data[i:j]`<br>`del obj.attr`<br>`del variable` |

# Оператор условия «if»

- Стиль C подобных языков

```
if (x > y) {
x = 1;
y = 2;
}
```

- Стиль python

```
if x > y:
    x = 1
    y = 2
```

# Оператор условия «if»

```
Строка заголовка:
    Вложенный блок операторов
```

- Круглые скобки необязательны:

```
if (x < y)        if x < y
```

- Конец строки является концом оператора

```
x = 1;          x = 1
```

# Оператор условия «if»

- Конец отступа является концом блока

```
if (x > y) {
  x = 1;
  y = 2;
}
```

```
if x > y:
  x = 1
  y = 2
Print(«end»)
```

# Синтаксическая модель Python:

- конец строки завершает оператор в этой строке (безо всяких точек с запятой);
- вложенные операторы объединяются в блок и ассоциируются согласно их физическим отступам (без фигурных скобок).

## Специальные правила для операторов:

- **Разделение операторов точкой с запятой**

```
a = 1; b = 2; print (a + b)
```

# Синтаксическая модель Python:

- Объединение операторов используя скобки

```
mylist = [1111,
          2222,
          3333]
```

```
X = (A + B +
     C + D)
```

```
X = A + B + \
    C + D
```

# Синтаксическая модель Python:

- **Специальные правила для блоков**

  тело составного оператора может взамен находиться в той же самой строке, что и строка заголовка оператора Python, после двоеточия:

```
if x > y: print (x)
```

# Интерактивные циклы

```
while True:
    reply = input('Enter text:’)
    if reply == 'stop’ : break
    print(reply.upper())
```

# Выполнение математических действий над пользовательским вводом

```
while True:
  reply = input('Enter text:')
  if reply == ’stop’: break
  print(int(reply) ** 2)
    print(’Bye’)
```

# Обработка ошибок путем проверки ввода

```
while True:
    reply = input('Enter text:')
    if reply == 'stop':
        break
    elif not reply.isdigit():
        print('Bad!' * 8)
    else:
        print(int(reply) ** 2)
print('Bye')
```

# Обработка ошибок с помощью оператора try

```
while True:
    reply = input('Enter text:')
    if reply == 'stop' : break
    try:
        num = int(reply)
    except:
        print(’Bad!' * 8)
    else:
        print(num ** 2)
print('Bye’)
```

# Поддержка чисел с плавающей точкой

```
while True:
    reply = input(’Enter text: ’)
    if reply == ’stop': break
    try:
        print(float(reply) ** 2)
    except:
        print('Bad!' * 8)
print('Bye')
```

# Операторы присваивания

- Присваивания создают ссылки на объекты.
- Имена создаются при первом присваивании
- Перед ссылкой именам должно быть выполнено присваивание.
- Некоторые операции неявно выполняют присваивания.

# Формы оператора присваивания

| Операция | Описание |
|---|---|
| `spam = 'Spam'` | Базовая форма |
| `spam, ham = 'yum', 'YUM'` | Присваивание кортежа (позиционное) |
| `[spam, ham] = ['yum', 'YUM']` | Присваивание списка (позиционное) |
| `a, b, c, d = 'spam'` | Присваивание последовательности, обобщенное |
| `a, *b = 'spam'` | Расширенная распаковка последовательности (Python 3.X) |
| `spam = ham = 'lunch'` | Групповое присваивание |
| `spams += 42` | Дополненное присваивание (эквивалентно `spams = spams + 42`) |

# Присваивание последовательности

```
>>> nudge = 1                       # Базовое присваивание
>>> wink = 2
>>> A, B = nudge, wink              # Присваивание кортежа
>>> A, B                            # Подобно A = nudge; B = wink
(1, 2)
>>> [C, D] = [nudge, wink]          # Присваивание списка
>>> C, D
(1, 2)
```

```
>>> [a, b, c] = (1, 2, 3)       # Присваивание кортежа значений списку имен
>>> a, c
(1, 3)
>>> (a, b, c) = "ABC"           # Присваивание строки символов кортежу
>>> a, c
('A', 'C')
```

# Присваивание последовательности

```
>>> a, b, c = string[0], string[1], string[2:]    # Индексация и нарезание
>>> a, b, c
('S', 'P', 'AM')
>>> a, b, c = list(string[:2]) + [string[2:]]     # Нарезание и конкатенация
>>> a, b, c
('S', 'P', 'AM')
>>> a, b = string[:2]                             # То же самое, но проще
>>> c = string[2:]
>>> a, b, c
('S', 'P', 'AM')
>>> (a, b), c = string[:2], string[2:]            # Вложенные последовательности
>>> a, b, c
('S', 'P', 'AM')
```

# Расширенная распаковка последовательностей в Python 3.X

```
C:\code> c:\python33\python
>>> seq = [1, 2, 3, 4]
>>> a, b, c, d = seq
>>> print(a, b, c, d)
1 2 3 4
>>> a, b = seq
ValueError: too many values to unpack (expected 2)
Ошибка значения: слишком много значений для распаковки (ожидалось 2)
```

```
>>> a, *b = seq
>>> a
1
>>> b
[2, 3, 4]
```

```
>>> *a, b = seq
>>> a
[1, 2, 3]
>>> b
4
```

```
>>> a, *b, c = seq
>>> a
1
>>> b
[2, 3]
>>> c
4
```

# Граничные случаи

```
>>> a, b, c, d, *e = seq
>>> print(a, b, c, d, e)
1 2 3 4 []
>>> a, b, *e, c, d = seq
>>> print(a, b, c, d, e)
1 2 3 4 []
```

```
>>> a, *b, c, *d = seq
SyntaxError: two starred expressions in assignment
```
*Ошибка синтаксиса: два имени со звездочкой в присваивании*

```
>>> a, b = seq
ValueError: too many values to unpack (expected 2)
```
*Ошибка значения: слишком много значений для распаковки (ожидалось 2)*

```
>>> *a = seq
SyntaxError: starred assignment target must be in a list or tuple
```
*Ошибка синтаксиса: имя со звездочкой должно находиться в списке или кортеже*

```
>>> *a, = seq
>>> a
[1, 2, 3, 4]
```

# Дополненные присваивания

```
X = X + Y                    # Традиционная форма
X += Y                       # Более новая дополненная форма
```

| | | | |
|---|---|---|---|
| X += Y | X &= Y | X -= Y | X \|= Y |
| X *= Y | X ^= Y | X /= Y | X >>= Y |
| X %= Y | X <<= Y | X **= Y | X //= Y |

```
>>> x = 1
>>> x = x + 1          # Традиционное присваивание
>>> x
2
>>> x += 1             # Дополненное присваивание
>>> x
3
```

# Дополненные присваивания

Дополненные присваивания обладают тремя преимуществами:

- Уменьшается объем набора на клавиатуре
- Левая сторона должна оцениваться только раз
- Оптимальная методика выбирается автоматически.

# Дополненные присваивания

```
>>> L = [1, 2]
>>> L = L + [3]       # Конкатенация: медленнее
>>> L
[1, 2, 3]
>>> L.append(4)       # Быстрее, но на месте
>>> L
[1, 2, 3, 4]
```

```
>>> L = L + [5, 6]          # Конкатенация: медленнее
>>> L
[1, 2, 3, 4, 5, 6]
>>> L.extend([7, 8])        # Быстрее, но на месте
>>> L
[1, 2, 3, 4, 5, 6, 7, 8]
```

```
>>> L += [9, 10]          # Отображается на L.extend([9, 10])
>>> L
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

# Правила именования переменных

- Синтаксис: (подчеркивание или буква) + (любое количество букв, цифр или подчеркиваний)
- Регистр символов имеет значение: SPAM - не то же самое, что и spam
- Зарезервированные слова не разрешены

| | | | | |
|---|---|---|---|---|
| False | class | finally | is | return |
| None | continue | for | lambda | try |
| True | def | from | nonlocal | while |
| and | del | global | not | with |
| as | elif | if | or | yield |
| assert | else | import | pass | |
| break | except | in | raise | |

# Соглашения по именованию

Имена, которые начинаются с одиночного подчеркивания (_X), не импортируются оператором from module import *

• Имена с двумя подчеркиваниями в начале и конце (__ X__ ) являются системными именами, которые имеют особый смысл для интерпретатора.

• Имена, начинающиеся с двух подчеркиваний, но не оканчивающиеся ими (__ X), локализованы (“искажены”) для включения в себя классов

• Имя, состоящее из одиночного подчеркивания (_), хранит результат последнего выражения при работе в интерактивном сеансе.

# Операции вывода

```
print([object, ...][, sep=' '][, end='\n'][, file=sys.stdout][, flush=False])
```

- sep — строка, вставляемая между текстовыми представлениями объектов, которой по умолчанию будет одиночный пробел, если она не задана; передача пустой строки полностью подавляет разделители.
- end — строка, добавляемая в конец выводимого текста, которой по умолчаниюбудет символ новой строки \n, если она не задана. Передача пустой строки позволяет избежать перехода на следующую строку вывода в конце выводимого текста — следующий вызов print продолжит добавление с конца текущей строкивывода.
- file указывает файл, стандартный поток данных или другой объект, подобныйфайлу, в который будет отправляться текст; если он не передан, то по умолчанию принимается поток стандартного вывода sys. stdout. Передавать можно любой объект, имеющий метод write (string), как у файловых объектов, но реальные файлы должны быть уже открыты для вывода.

# Функция print

```
>>> print()                          # Отображение пустой строки
>>> x = 'spam'
>>> y = 99
>>> print(x, y, z, sep='')      # Подавить разделитель
spam99['eggs']
>>>
>>> print(x, y, z, sep=', ')    # Специальный разделитель
spam, 99, ['eggs']
```

ментов

```
>>> print(x, y, z, sep='')      # Подавить разделитель
spam99['eggs']
>>>
>>> print(x, y, z, sep=', ')    # Специальный разделитель
spam, 99, ['eggs']
```

# Функция print

```
>>> print(x, y, z, end='')     # Подавление разрывов строк
spam 99 ['eggs']>>>
>>>
>>> print(x, y, z, end=''); print(x, y, z)   # Два оператора print,
                                             #   выводящие в одну строку
spam 99 ['eggs']spam 99 ['eggs']
>>> print(x, y, z, end='...\n')              # Специальный конец строки
spam 99 ['eggs']...
>>>

>>> print(x, y, z, sep='...', end='!\n')     # Множество ключевых аргументов
spam...99...['eggs']!
>>> print(x, y, z, end='!\n', sep='...')     # Порядок не имеет значения
spam...99...['eggs']!
```

# Функция print

```
>>> print(x, y, z, sep='...', end='!\n')     # Множество ключевых аргументов
spam...99...['eggs']!
>>> print(x, y, z, end='!\n', sep='...')     # Порядок не имеет значения
spam...99...['eggs']!
```

```
>>> text = '%s: %-.4f, %05d' % ('Result', 3.14159, 42)
>>> print(text)
Result: 3.1416, 00042
>>> print('%s: %-.4f, %05d' % ('Result', 3.14159, 42))
Result: 3.1416, 00042
```