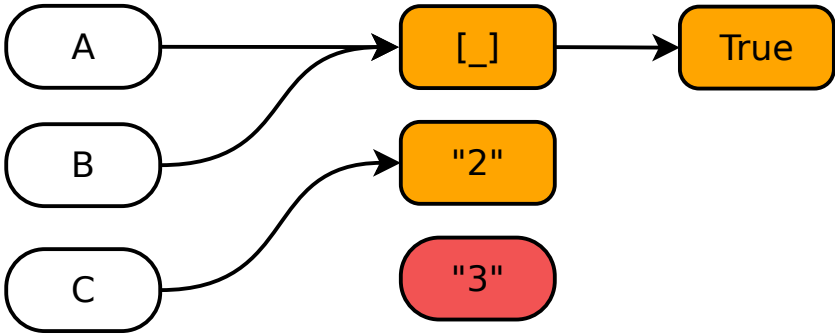
Переменные (идентификаторы)

- Переменные - имена для объектов (ссылки на объект)
- Имя - [a-zA-Z_][a-zA-Z_0-9]*
- Переменная создается присваиванием
  name = val (a = b = 1 работает, но a = (b = 1) нет)
- У одного объекта может быть много имен
- Часть имен зарезервирована (for, while, in, ..)
  import keyword; print keyword.kwlist
- Типизация динамическая - тип связывается с объектом. Можно считать что все переменные имеют тип (PyObject *)
- В компилируемых языков другой подход. Переменная как ящик, в который кладутся данные и переменная определяет их поведение (тип)



last 3 images by David Goodger
http://python.net/ goodger/projects/pycon/2007/idiomatic/handout.html

Базовые типы данных

- None
- int (C long int), long
- float (c double)
- str, unicode
- bool – True, False
- complex
- ...

Операции

- Все базовые типы данных - константны. Любая операция создает новый объект
- + - / * % // **
- Нет --, ++ (Вместо этого -= 1, += 1)
- +=, -=, *=, ...
- >, <, != (<> - устарел), >=, <=, == Не сравнивайте разные типы данных
- is vs ==, is not
- or and not
- & | ^
- 0 < x == y < 10
- math, cmath

Базовые типы данных

- Одно и то-же имя(переменная) может в разные моменты указывать на разные типы данных

```
x = 12
x = "12"
```

- Строгая типизация - не производится автоматических приведений типов, кроме очевидных (int -> float)

```
1 + "2" # error
1 + 1.0 == 2.0
(-1) ** 0.5 # error - no float -> complex conversion
```

- Однако все приводится к bool при использовании операций and/or/not

```
Empty objects, 0, 0.0, 0 + 0j, None => False
All else => True
obj and obj
obj or obj
```

- Типы нужно приводить явно

```
int("12") == 12
str(12) == "12"
```

```
3           repr("12") == "12"
4           1 + int("2") == 3
```

Строки

```
1       "abc"
2       'abc'
3
4       b'abc'
5       B'abc'
6
7       """abcdef
8       gj
9       h""" == "abcdef\ngj\nh"
10
11      r"C:\temp\dir\fname" == "C:\\temp\\dir\\fname"
12
13      U"Unicode␣text"
14      RU"Raw␣unicode␣text"
```

Строковые операции

В python функции/методы возвращающие значение - делают копию, возвращающие None - модифицируют объект на месте.

```
1   "abc" + "def" == "abcdef"
2
3   # a * N – a + a + a + ... + a, N раз
4   "ab" * 3 == "ababab"
5
6   "abcdef"[3] == "d"
7
8   # a[x:y] – substring [x, y)
9   "0123456789"[2:4] == "23"
10
11  a.replace(from, to)
12  "x␣+␣y".replace("+", "//") == "x␣//␣y"
13
14  a.find(string, pos)
15  "abcd".find("cd") => 2
16
17  # for x[:x.index(substr)]
18  "abcd".find("t") => -1
19
20  "cdabcd".find("cd", 1) => 4
```

```
21
22      a.index(string)
23      "abc".index("fd") # error
24
25      a.split(string)
26      "a,b,c,d".split(",") == ("a", "b", "c", "d")
27
28      "abc".startswith("ab") == True
29      "abc".endswith("dabc") == False
```

Строки форматирование

- %, format

```
1     "size␣=␣%d␣%s" % (12, 'cm')
2     # "size = 12 cm"
3
4     "size␣=␣%(sz)d␣%(units)s" % dict(sz=1, units='m')
5     # "size = 1 m"
6
7     "Some␣{}␣{}".format("very␣interesting", "text")
8     #"Some very interecting text"
9
10    "Brown␣fox␣{what}␣over".format(what="jump")
11    #"Brown fox jump over"
12
13    #"{name or index or EMPTY!conversion:format_spec}"
14    "{:␣^25}".format(xx) == '␣␣␣␣␣␣␣␣␣␣␣␣xx␣␣␣␣␣␣␣␣␣␣␣␣'
```

Unicode

- Unicode vs encoding
- Unicode libraries
- Unicode libraries complexity and problems
- encode/decode/encoding module

type & id & hash & isinstance

- type(x) => тип x
- id(x) – int идентификатор значения (адрес в памяти)
- hash(x) - int - хэш значения, широко используется внутри питона
- isinstance(x, X) - проверяет, что x имеет тип X isinstance(1, int) X может быть списком в скобках

```
1     type(1) == int
2     d = "asad"
3     type(d) == str
4     type(1 is 2) == bool
5
6     id(1) == 34564790 #example
7     a is b # same as id(a) == id(b)
8
9     hash(1) == 1
10    hash("1") == 1977051568
```

Особенности сравнений

- Сравнения между разными типами данных работает, но смысла не имеет
- Кроме комплексных чисел
- И юникодных строк с бинарными (иногда, зависит от кодировки - sys.getdefaultencoding())

```
1    1 > "d" # False
2    1 > 1 + 1j # error
3    "d" > 2 + 1j # True
```

AI

- Прочитать описание строковых операций
- Прочитать описание format
- Посмотреть модули math/cmath с помошью ipython
- Посмотреть приоритет операций