Introduction to Python

ELTE - Budapest- September 2022

Tutorial Outline

- interactive "shell"
- basic types: numbers, strings
- container types: lists, dictionaries, tuples
- variables
- control structures
- functions & procedures
- classes & instances
- modules & packages
- exceptions
- files & standard library

Interactive “Shell”

- Great for learning the language
- Great for experimenting with the library
- Great for testing your own modules
- Two variations: IDLE (GUI),
  python (command line)
- Type statements or expressions at prompt:

```
>>> print "Hello, world"
Hello, world
>>> x = 12**2
>>> x/2
72
>>> # this is a comment
```

Numbers

- The usual suspects
  - 12, 3.14, 0xFF, 0377, (-1+2)*3/4**5, abs(x), 0<x<=5
- C-style shifting & masking
  - 1<<16, x&0xff, x|1, ~x, x^y
- Integer division truncates :-(
  - 1/2 -> 0 # 1./2. -> 0.5, float(1)/2 -> 0.5
  - Will be fixed in the future
- Long (arbitrary precision), complex
  - 2L**100 -> 1267650600228229401496703205376L
    - In Python 2.2 and beyond, 2**100 does the same thing
  - 1j**2 -> (-1+0j)

Strings

- "hello"+"world" "helloworld" # concatenation
- "hello"*3 "hellohellohello" # repetition
- "hello"[0] "h" # indexing
- "hello"[-1] "o" # (from end)
- "hello"[1:4] "ell" # slicing
- len("hello") 5 # size
- "hello" < "jello" 1 # comparison
- "e" in "hello" 1 # search
- "escapes: \n etc, \033 etc, \if etc"
- 'single quotes' """triple quotes""" r"raw strings"

Lists

- Flexible arrays, *not* Lisp-like linked lists
  - a = [99, "bottles of beer", ["on", "the", "wall"]]
- Same operators as for strings
  - a+b, a*3, a[0], a[-1], a[1:], len(a)
- Item and slice assignment
  - a[0] = 98
  - a[1:2] = ["bottles", "of", "beer"]

    -> [98, "bottles", "of", "beer", ["on", "the", "wall"]]
  - del a[-1] # -> [98, "bottles", "of", "beer"]

More List Operations

```
>>> a = range(5)          # [0,1,2,3,4]
>>> a.append(5)           # [0,1,2,3,4,5]
>>> a.pop()               # [0,1,2,3,4]
5
>>> a.insert(0, 42)       # [42,0,1,2,3,4]
>>> a.pop(0)              # [0,1,2,3,4]
5.5
>>> a.reverse()           # [4,3,2,1,0]
>>> a.sort()              # [0,1,2,3,4]
```

Dictionaries

- Hash tables, "associative arrays"
  - d = {"duck": "eend", "water": "water"}
- Lookup:
  - d["duck"] -> "eend"
  - d["back"] # raises KeyError exception
- Delete, insert, overwrite:
  - del d["water"] # {"duck": "eend", "back": "rug"}
  - d["back"] = "rug" # {"duck": "eend", "back": "rug"}
  - d["duck"] = "duik" # {"duck": "duik", "back": "rug"}

More Dictionary Ops

- Keys, values, items:
  - d.keys() -> ["duck", "back"]
  - d.values() -> ["duik", "rug"]
  - d.items() -> [("duck","duik"), ("back","rug")]
- Presence check:
  - d.has_key("duck") -> 1; d.has_key("spam") -> 0
- Values of any type; keys almost any
  - {"name":"Guido", "age":43, ("hello","world"):1, 42:"yes", "flag": ["red","white","blue"]}

Dictionary Details

- Keys must be **immutable**:
  - numbers, strings, tuples of immutables
    - these cannot be changed after creation
  - reason is *hashing* (fast lookup technique)
  - **not** lists or other dictionaries
    - these types of objects can be changed "in place"
  - no restrictions on values
- Keys will be listed in **arbitrary order**
  - again, because of hashing

Tuples

- key = (lastname, firstname)
- point = x, y, z # parentheses optional
- x, y, z = point # unpack
- lastname = key[0]
- singleton = (1,) # trailing comma!!!
- empty = () # parentheses!
- tuples vs. lists; tuples immutable

Variables

- No need to declare
- Need to assign (initialize)
  - use of uninitialized variable raises exception
- Not typed

```
if friendly: greeting = "hello world"
else: greeting = 12**2
print greeting
```

- ***Everything*** is a "variable":
  - Even functions, classes, modules

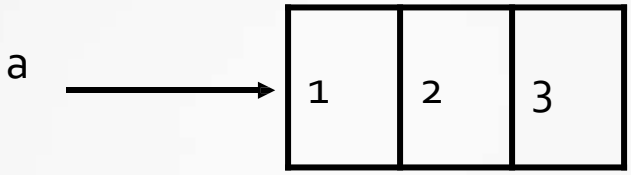
Reference Semantics

- Assignment manipulates references
  - x = y **does not make a copy** of y
  - x = y makes x **reference** the object y references
- Very useful; but beware!
- Example:

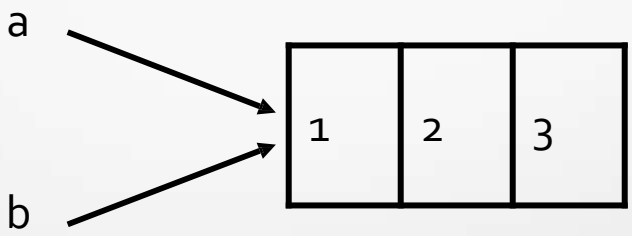
```
>>> a = [1, 2, 3]
>>> b = a
>>> a.append(4)
>>> print b
[1, 2, 3, 4]
```

Changing a Shared List

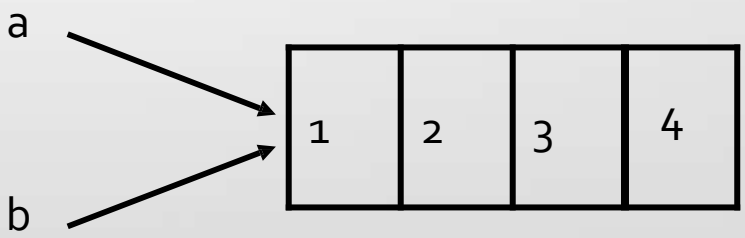
a = [1, 2, 3]



b = a



a.append(4)

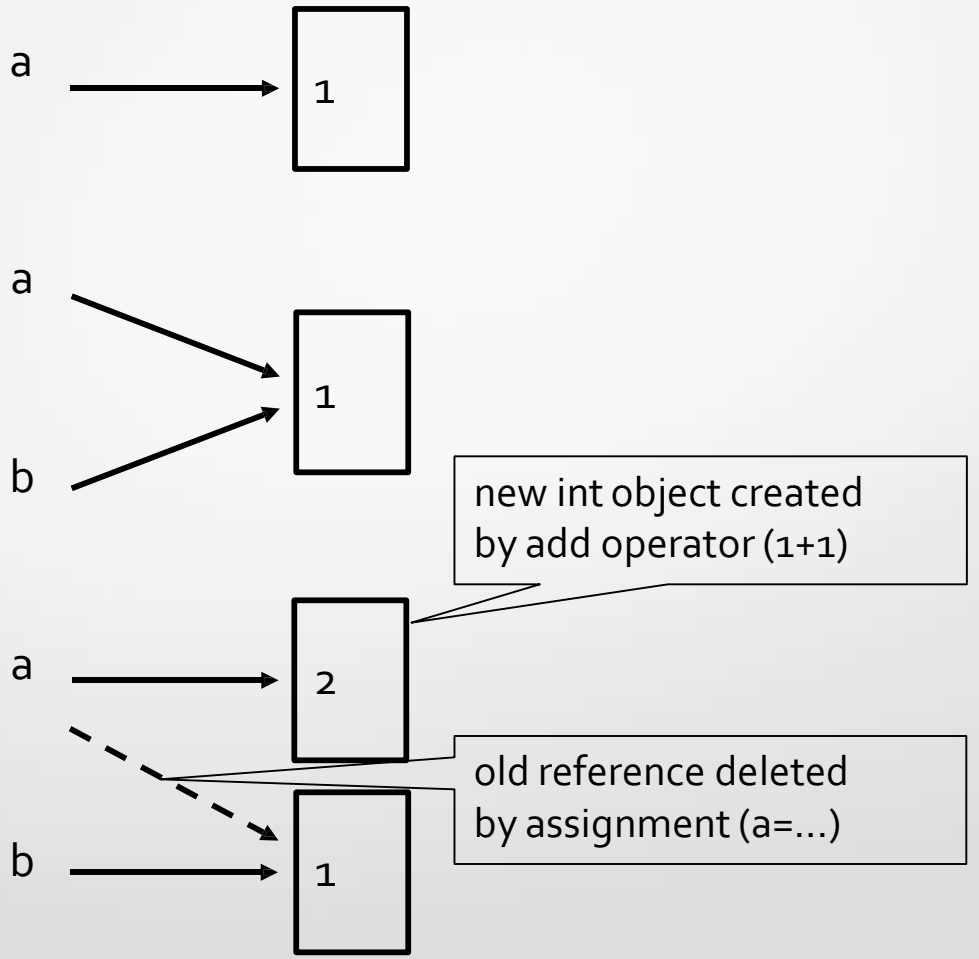


Changing an Integer

a = 1

b = a

a = a+1



Control Structures

if *condition*:

    statements

[elif *condition*:

    statements] …

else:

    statements

while *condition*:

    statements

for *var* in *sequence*:

    statements

break

continue

Grouping Indentation

In Python:

```
for i in range(20):
    if i%3 == 0:
        print i
        if i%5 == 0:
            print "Bingo!"
    print "---"
```

In C:

```
for (i = 0; i < 20; i++)
{
    if (i%3 == 0) {
        printf("%d\n", i);
        if (i%5 == 0) {
            printf("Bingo!\n"); }
    }
    printf("---\n");
}
```

```
0
Bingo!
---
---
---
3
---
---
---
6
---
---
---
9
---
---
---
12
---
---
---
15
Bingo!
---
---
---
18
---
---
```

Functions, Procedures

```
def name(arg1, arg2, ...):
    """documentation"""       # optional doc string
    statements

return                  # from procedure
return expression       # from function
```

Example Function

```
def gcd(a, b):
   "greatest common divisor"
   while a != 0:
      a, b = b%a, a    # parallel assignment
   return b

>>> gcd.__doc__
'greatest common divisor'
>>> gcd(12, 20)
4
```