Script language: Python

Data structures

Cédric Saule

Technische Fakultät
Universität Bielefeld

10. Februar 2016

Immutable vs. Mutable

Previously known types: `int` and `string`.

- Both are Immutable $\rightarrow$ but what does it mean?
  - Numbers are fix and cannot be modified.
  - New allocation by: `number = number + 1`
  - When assigning, a new object is created with the desired value.
- So what are Immutable for?
  - Mutable (variable data types) use Immutable for addressing.
  - Be referred to as *hashable*.
  - Same Immutable reference (in part) to optimize memory on the same memory address.

Data types in Python.

There is different „*built-in*“ (`immutable`, *mutable*) data type:

Logical values `bool`

Numbers `int`, (`long`), `float`, `complex`

Sequential data `string`, `tuple`, *list*

Mapping *dict*

Sets *set*, `frozenset`

Special cases: `None`.

- “Worth „nothing
- Met in:

Java `null` Perl `undef`

Python

–

Logical

Boolean, Truth tests, equality operators.

What are logical values in Python?

- Logical values are clearly defined by `bool`.
  - `True`.
  - `False`.
- In addition, each truth test result is an object `True`.

What are logical values in Python?

The following statements are considered as `False`:

None	
False	
0, 0L, 0.0, 0j	the number 0
, '', (), []	Empty sequential data type
{}	Empty mapping.

Special case for some classes which implement one of the following methods and would provide this as a return value: `False` or `0`.

- `__nonzero__()`
- `__len__()`

Boolean operators

The following operators replace the usual shortcuts `||`, `&&` `!`, respectively sorted by order of priority. In addition, these operators are also special functions which are their own input value and have the following outputs:

X or Y	logical *OR*, as a function: If `X == False`, then `Y`, otherwise `X`
X and Y	logical *AND*, as a function: If `X == True`, then `Y`, otherwise `X`
not X	logical negation, (no `!`; Negate the statement fowarded by the command line.)

Comparison operators

- can be concatenated:
  `X < Y <= Z` is equivalent to `X<Y and Y<=Z`, where the last expression is evaluated only if the first is evaluated as `True`.
- All objects can be compared.
- Objects of different types cannot be equal but they are always sorted in the same order $\rightarrow$ so, in a heterogeneous array, they will be sorted on the same order.
- Objects can be evaluated as equal by implementing `__cmp__()` but they must be of the same type.

Comparison operators

`<`	smaller than
`>`	greater than
`==`	equal
`is`	objects identity
`<=`	smaller or equal
`>=`	greater or equal
`!=`	different
`is not`	object with different identities.

- <, <=, > and >= throw `aTypeError` when we compare complex numbers.
- With sequential data types, it exists two other comparison operators.

`in`	is included.
`not in`	is not included.

Python

–

Numbers

Integer, Float, Complex

Numbers

Four different types of numbers exist in Python 2.X.

Integer/Long Integers

```
100          # Integer
100L         # Long
sys.maxint   # Maximal size for integers.
             # Bigger ones are
    # automatically  converted
             # into long.
```

Float Floating point numbers, `1.` or `-55.3`

Complex Complex numbers, `1.5+2j`

Numbers – Special & Operations

- Since the Python 2.6 and 3, Integers and Longs have been merged.
- When a range number is too large for an Integer, it is automatically converted into Long.
- `int <OPERAND> int = int` and `float <OPERAND> int = float`

+, -, *, / ... ||

// Integer portion of the division.

% Modulo

-X Opposite of X

+X Unchanged X

Y**Z Y^Z

Shorthand: `x <OPERAND>= y` $\rightarrow$ `x = x <OPERAND> y`

Numbers – Important functions

abs()	Absolute value
long()	Output as `long`
complex(r, i)	Complex number, (`r`)Real part, (`i`)Imaginary part
divmod(x,y)	`(x // y, x % y)`
int()	Output as `int`
float()	Output as `float`
c.conjugate()	Computes the conjugate of a complex number
math.*	See `math`-Module

Exercises – Operators precedence

Compute the following operations. What is the the operators precedence?

```
> 4 * 3 + 6
> 4 + 3 * 6
> 100 / 5 % 3
> 5 * 6 ** 7
> res = 7.
> n = 5
> res //= -n
```

Python

–

Sequential data types.

String, Tuple, List

Sequential data types

Sequential data types can generally be seen as containers for sequences of objects and their references. For Strings, Lists and Tuples, we apply the same standard access operators.

```
> s = "Hello you!"
      0123456789
```

```
> s[2:7]
  'llo y'
```

```
> s[0]
  'H'
```

```
> list = [s, 54, 'World']
         0   1    2
```

```
> list[1]
  54
```

```
> list[2][0:2]
  'Wo'
```

Sequential data types – Operations & Functions

x in s	`True`, if `x` is in `s`, otherwise `False`.
s*n, n*s	Concatenates `n` copies of `s`.
s[i]	Object in the `i`'th position.
s[i:j]	Slice from `i` to `j`.
len(s)	Size of `s`.
s.index(i)	First occurrence of `i` in `s`
x not in s	`False`, if `x` is in `s`, otherwise `True`.
s+t	Concatenation of `s` and `t`.
s[i:j:k]	Slice from `i` to `j` with an increase of `k`.
min(s)	Smallest element in `s`.
max(s)	Greatest element in `s`.
s.count(i)	Number of all `i` in `s`.

String

- String of any length.
- Strings are instanciated between "␣" or '␣'
- String indices begin with 0 and the last index is: `len(string)-1`.
- *Immutable* → Change of form `s[0] = "a"` not allowed.
- Escape character: „\ “.
- Only ASCII characters can be processed easily into 2.x.
  - String != Unicode
  - In Python 3.x, both types were merged for the first time.

```
s = "I␣am␣a␣string␣of␣size␣24"
```

String – Important methods

str(<INPUT>) `INPUT` as string.

count(sub[, start[, end]]) Number of non-overlapping substrings `sub`.

endswith(suffix[, start[, end]]) `True`, if the string finishes with `suffix`, then `False`; Can also be a String with a `tuple`.

find(sub[, start[, end]]) Index Position of the first occurence of `sub`, returns `-1` otherwise.

r/lstrip([chars]) Removes the string `chars` from `left` or `right`, no parameters or `None` space. If you want to strip on both sides, use `strip([chars])`.

- Specify optional `start` and `end` in *slice* notation.

String – Important methods

partition(sep) The string is split at `sep`; Returns a 3-uple: `(prefix, sep, suffix)`, if `sep` is not found, we will have `(string,'','')`.

split([sep[, maxsplit]]) Split the String after each `sep` if it is given, otherwise after each white space. `maxsplit` specify the maximal number of plit parts.

capitalize() The first letter is in capital, the others are in small.

swapcase() Small letters become capital and vice versa.

join(<ITERABLE>) starts all elements from `ITERABLE` to the String.

Exercise – String and numbers

A first small exercise on the theme of types, conversions and modifications:

1. Create the number `z` with the value `4353,3`.
2. Convert it into `string` and split it by the point and save the result in your own variable `l`.
3. Add `1` to the integer part and save it again as `string` in `l`.
4. Overwrite `z`, now it should be in the front of the decimal point and the modified integer value of `l` should be behind the point.

`z` and `l` and their contents should always be of the same type!

Tuple

- No changeable list $\rightarrow$ *Immutable*.
- Any length.
- Elements can be of different types.
- Notation:
  - Content is between parentheses `(...)`.
  - Listed objects will be comma-separated.

```
> s1 = (u"are" ,1)
> suffix =  "tuple"
> _tuple = (s1, 1337, suffix)
> _tuple
  ((u"are", 1), 1337, "tuple")
```

List

- Changeable list $\rightarrow$ *Mutable*.
- Any length.
- Elements can be of different types.
- Notation:
  - Content is between square brackets `[...]`.
  - Listed objects will be comma-separated.

```
> s1 = (u"are" ,1)
> suffix =  "tuple"
> _list = [s1, 1337, suffix]
> _list[2] = "list"
> _list
  [(u"are", 1), 1337, "list"]
```

Exercise – (Im)mutable

1. Create a variable `tuple` as seen on the tuple slides.
2. Instanciate a variable of type `list` with the content of the `_tuple`. Modify the list so that its content be the same as `_list`.
3. Convert back into the first list in tuple and set the content once again in a new variable of `tuple`.

Use the functions `list()` as well as `tuple()`.

Operations on sequential *mutable* data types.

Replace and remove references:

s[i] = x	Reallocation.
s[i:j] = t	Slice replacement with *iterable* `t`.
s[i:j:k] = t	Interval increased with `k` with the elements of `t`.
s.insert(i, x)	`x` an index `i` slide $\rightarrow$ `s[i:i] = [x]`.
del s[i(:j(:k))]	Drop elements, `j` and `k` are optional.

Operations on sequential *mutable* data types.

Find references, change data structures:

s.index(x[,i[,j]]) Index of the elements `x`
$\rightarrow$ `s[k] == x` and `i <= k < j`.

s.remove(x) `x` is deleted from `s` $\rightarrow$ `del s[s.index(x)]`.

s.extend(x) Adds all element of `x` to `s`.

Exercises – Lists

Create a List with these elements: `1, 2, 3, 4, 5, 6, 7, 8, 9`

- Remove the first element.
- Remove the first element and add it again to the list.
- Remove the second and third element.
- Replace the forth element with the size of the list.
- Append the list with `[10, 11, 12]` at the end of the list.
- Append the list with `[10, 11, 12]` between the elements 7 and 8.

Give each step from the list.

Operations on sequential *mutable* data types.

List as *Queue*:

s.append(x) Adds `x` to `s` → `s[len(s):len(s)] = [x]`.

s.pop([i]) Delete the last elements of the list and return them →
`x = s[i]; del s[i]; return x`.

```
> list = []
> store = list.append
> store('apple')
> store(math.pi)
> list
  ['apple', 3.141592653589793]
```

Operations on sequential *mutable* data types.

Change objects order:

s.reverse() Reverse elements order.

s.sort([cmp[, key[, reverseP]]]) ...

- Both methods change the sequential data structure directly
  - No returned value.
  - Memory efficient.
- Particularity of `sort(...)`: Details for `cmp`-Function in *Advanced data structures*.

Exercises – Reverse strings

Write a short script that reverses a given string. So `Here I am!` would become `!ma I ereH`.

Python

–

Mapping data types.

Dict[ionary] (Hashmap)

Mapping data types. – dict

- Until now, `dict` is the only mapping data type in Python.
- This data type is *mutable*.
- It consists of unordered pairs of `key : value`.
  - `key` must be hashable $\rightarrow$ *immutable*!!!
  - `value` can be of any type.
  - Caution when you use numbers as `key`.
    - Equivalent point to the same `value`: `1 == 1`.
    - `float` very inappropriate (**floating** point number!!11ELF.)
- Standard notation:

  ```
  dict1 = {key0: value0, key1: value1, key2: value2}
  ```

- Specified in standard notation or on `dict`'s constructor.

Dictionary creation

dictionaries can be instanciated by the standard notation or will be instanciated through the call of the following constructors.

```
class dict(**kwarg)
class dict(mapping, **kwarg)
class dict(iterable, **kwarg)
```

Example:

```
> a = dict(one=1, two=2)
> b = {'one': 1, 'two': 2}
> c = dict(zip(['one', 'two'], [1, 2]))
> d = dict([('two', 2), ('one', 1)])
> e = dict({'one': 1, 'two': 2})
> a == b == c == d == e
True
```

Read and modify a dictionnary.

A `value` can be directly accessed by its `key` (Read, Modification, Addition of new mappings):

```
> b = {1: 'on', 2: 'two'}
> b[1]
  'on'

> b[1] = 'one'
> b[1]
  'one'

> b[3] = 'three'
> b
  [2; 'two', 3: 'three', 1: 'one'}
```

Python

–

Set Data types

Mix – Set, Frozenset

Set, Frozenset – Mix

`set` and `frozenset` are containers for unique element.
Mathematically, they can be considered as sets.

- Each object (element) must be *immutable*.
- There is some methods to specific mixing: `union()`, `intersetion()`, `issubset()`,`issuperset()`,`isdisjoint()`, ...

We give two different implementations for mixing:

set	Variable mix
frozenset	Non-variable mix $\rightarrow$ immutable

Set, Frozenset – Example of code

```
> s = set('abc')
> s
  {'a', 'b', 'c'}

> s.update('x', [8])
> s
  {8, 'a', 'b', 'c', 'x'}

> s.intersection('ac8')
  {'a', 'c'}
```

Set, Frozenset

- Constructors: `class set([iterable])`, `class frozenset([iterable])`.
- Short hands-Instantiation without constructor: `lem1, elem2, ..., elemNelem1, elem2, ..., elemN`.
- In sets of sets the referenced set type must be a `frozenset`.

Other methods and explanations: Python 2.7-Documentation