Strings, Tuples, Lists compound data types traversing them

Tomas Svoboda
http://cmp.felk.cvut.cz/~svoboda
Programming Essentials, EECS, CTU in Prague
2016-10-27

compound data

- a string consists of characters
- access as a whole (one variable)
- access individual elements

accessing the whole

```python
greetings = "Hello students"
new_greetings = greetings.swapcase()
print(greetings, new_greetings)
```

1. a new string variable
2. a string *method* creates a new string
3. just a print to standard output

accessing elements

```
greetings = "Hello students!”
first_char = greetings[0]
```

- how to get the last character?
- how to get the first word?

lenght of a string

```python
greetings = "Hello students!"
number_of_characters = len(greetings)
```

```
>>> greetings[number_of_characters]
```

string traversal - using index

```python
greetings = "Hello students!"
i = 0
while i<len(greetings):
    print(greetings[i])
    i += 1
```

string traversal - for loop

```python
greetings = "Hello students!"
for c in greetings:
    print(c)
```

for with indexes

```python
greetings = "Hello students!"
for idx,c in enumerate(greetings):
    print(idx,c)
```

slices

```python
greetings = "Hello students!"
substr1 = greetings[0:5]
substr2 = greetings[6:]
print(substr1)
print(substr2)
```

comparisons

- `==`
- `>, <`
- …

strings are *immutable*

```python
greetings = "Hello students!"
greetings[-1] = "?"
print(greetings)
```

`in` and `not in` operators

```python
greetings = "Hello students!"
if "!" in greetings:
    print("do not shout!")
```

how to remove special characters?

```python
special_chars = " !?"
greetings = "?Hello students!"
new_greetings = ""
for c in greetings:
    if c not in special_chars:
        new_greetings += c

print(greetings)
print(new_greetings)
```

Tuples

Tuples (value1, value2, ..)

- indexes
- slices
- immutable
- parentheses not strictly required, but think about readability

Tuple assignment

- packing
- unpacking

Lists

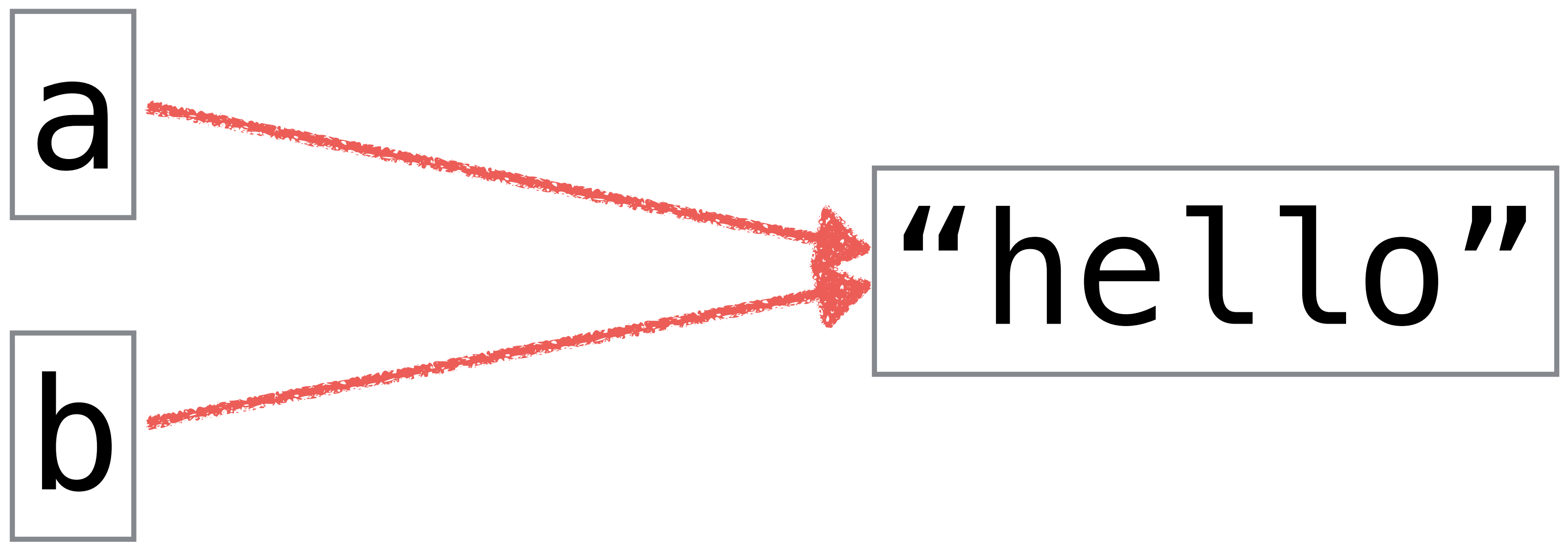
`[item0, item1, …]`

- similar to strings, tuples, **but**
- lists are mutable - we can change individual items

Objects, references, aliasing

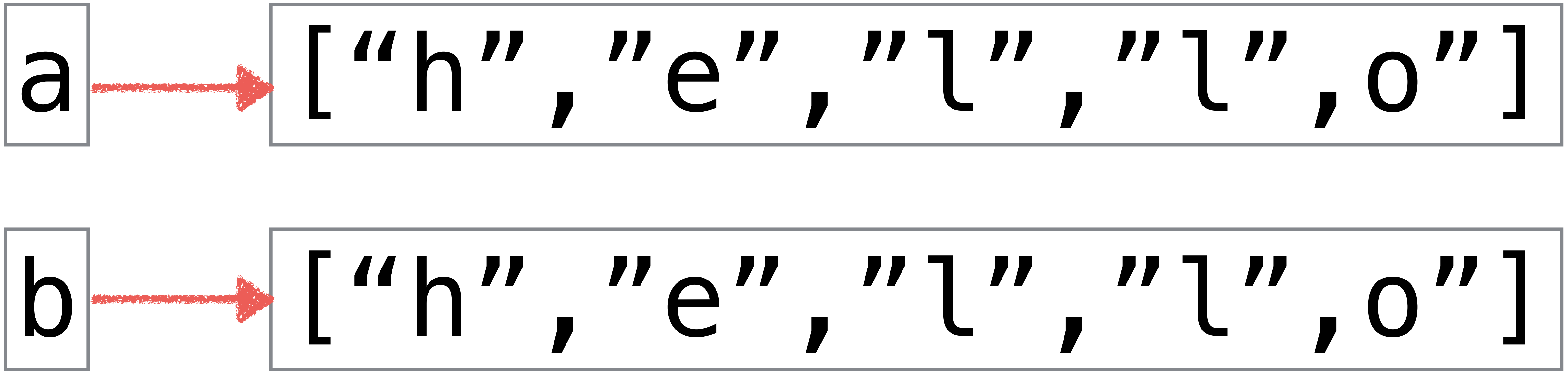
immutable objects

```
a = "hello"
b = "hello"
print(a == b)
print(a is b)
```



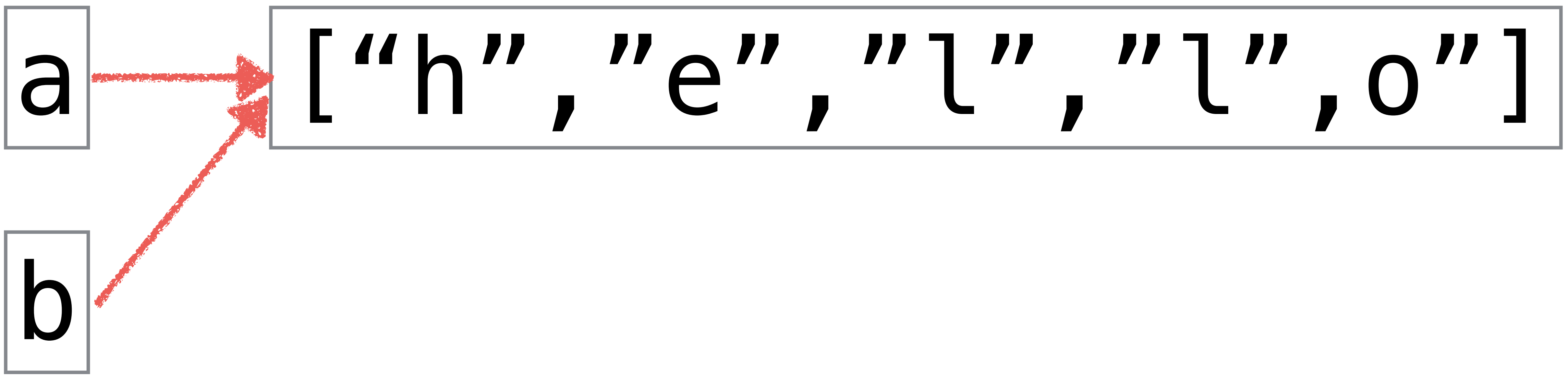
mutable objects

```python
a = ["h","e","l","l","o"]
b = ["h","e","l","l","o"]
print(a == b)
print(a is b)
```



aliasing

```python
a = ["h","e","l","l","o"]
b = a
print(a == b)
print(a is b)
```



cloning

```python
a = ["h","e","l","l","o"]
b = a[:]
print(a == b)
print(a is b)
```

List as an argument

- `def scale_values(input_list):`
- only a reference (pointer to the data) is passed not a clone/copy

pure functions and modifiers

- it is about semantics, not syntax
- pure functions communicates with the caller only through parameters (think about math functions)
  - do not alter the input parameters
  - create/compute a new data/variable and return reference to it
- function-modifiers
  - modify the input parameters/arguments
  - or have other side effects (printing, sending emails …)

functions that produce lists

- `def fcn(par):`
- `initialize result as empty list`
- `loop`
    - `create a new element`
    - `add to the result`
- `return result`

`list`

- `list(iterable)`
- creates a list from any iterable (string, list, generator …)

`range`

- `range(0,100)`
- does not compute all values instantly
- returns next when needed
- `list(range(0,100))`
- useful in for loops

nested lists

- element of a list can be anything …
- also other list
- think about matrices

summary

- compound types (elements, the whole unit)
- immutable/mutable
- reference, clone (`==, is`)
- pure functions vs. modifiers