

# Strings, Tuples, Lists compound data types

Tomas Svoboda

<http://cmp.felk.cvut.cz/~svoboda>

Programming Essentials, EECS, CTU in Prague

# compound data

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

# accessing the whole

```
1 greetings = "Hello students"  
2 new_greetings = greetings.swapcase()  
3 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

```
1 greetings = "Hello students!"  
2 first_char = greetings[0]
```



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

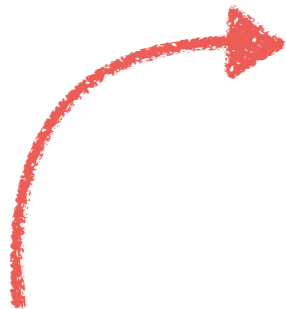
# length of a string

```
1 greetings = "Hello students!"  
2 number_of_characters = len(greetings)
```

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

# string traversal - using index

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



# string traversal - for loop

```
1 greetings = "Hello students!"  
2 for c in greetings:  
3     print(c)
```

# slices

```
1 greetings = "Hello students!"
2 substr1 = greetings[0:5]
3 substr2 = greetings[6:]
4 print(substr1)
5 print(substr2)
```



# comparisons

- $==$

- $>$   $<$

- $...$

# strings are *immutable*

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

# in and not in operators

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

how to remove special characters?

```
1 special_chars = " !?"
2 greetings = "?Hello students!"
3 new_greetings = ""
4 for c in greetings:
5     if c not in special_chars:
6         new_greetings += c
7
8 print(greetings)
9 print(new_greetings)
```

# Tuples

# Tuples (value1, value2, ..)

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

# Tuple assignment

- packing
- unpacking

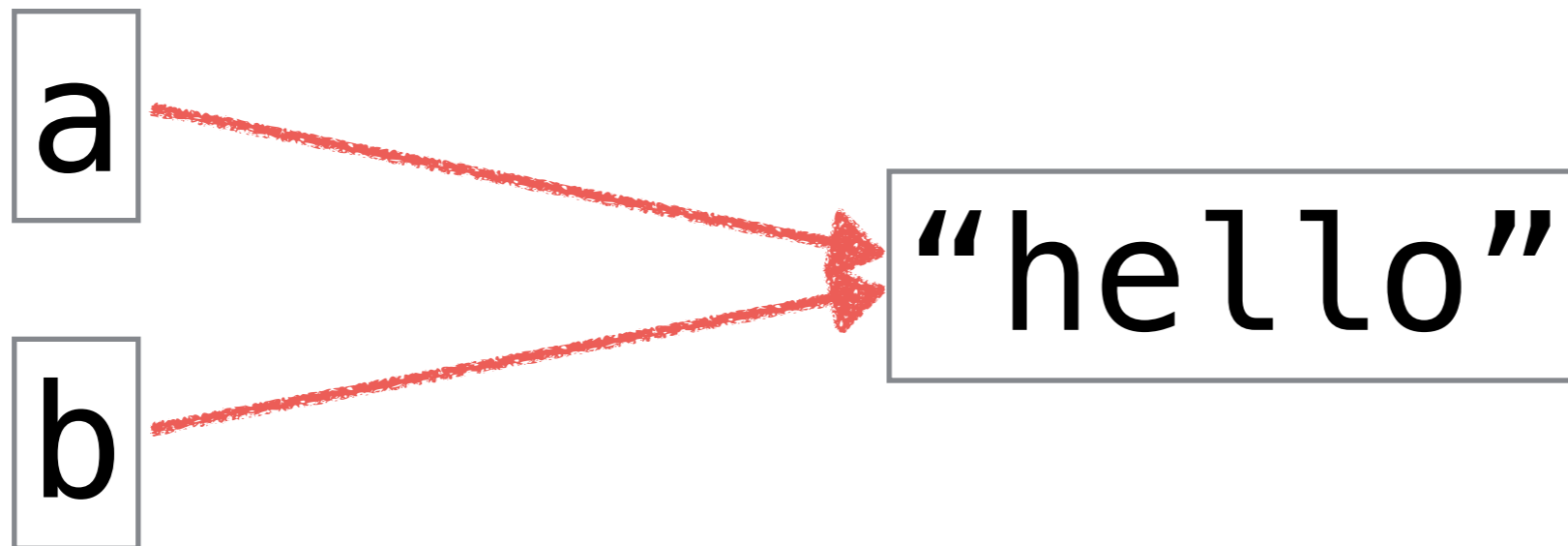
# Lists



# Objects, references, aliasing

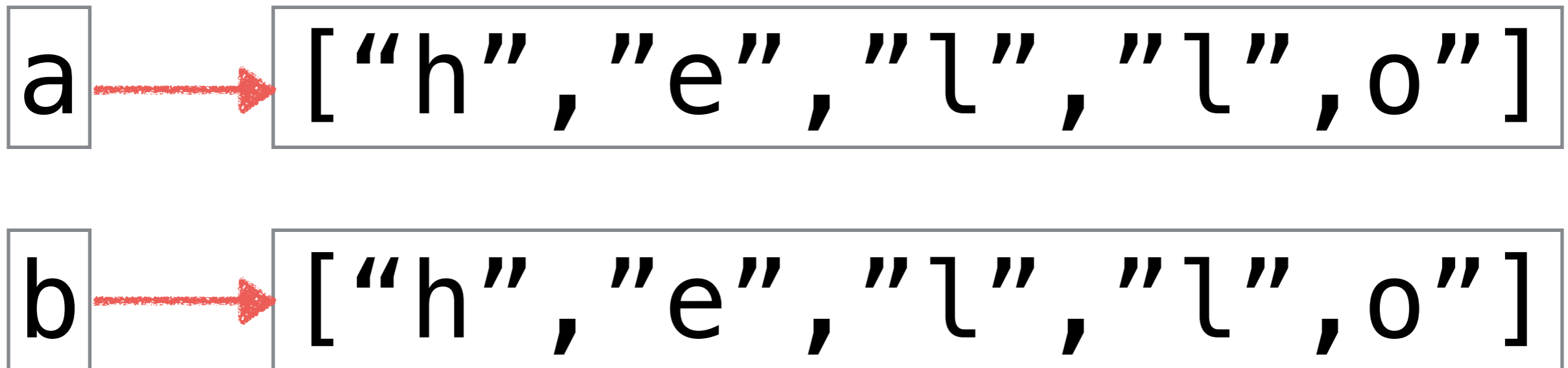
# immutable objects

```
1 a = "hello"  
2 b = "hello"  
3 print(a == b)  
4 print(a is b)
```



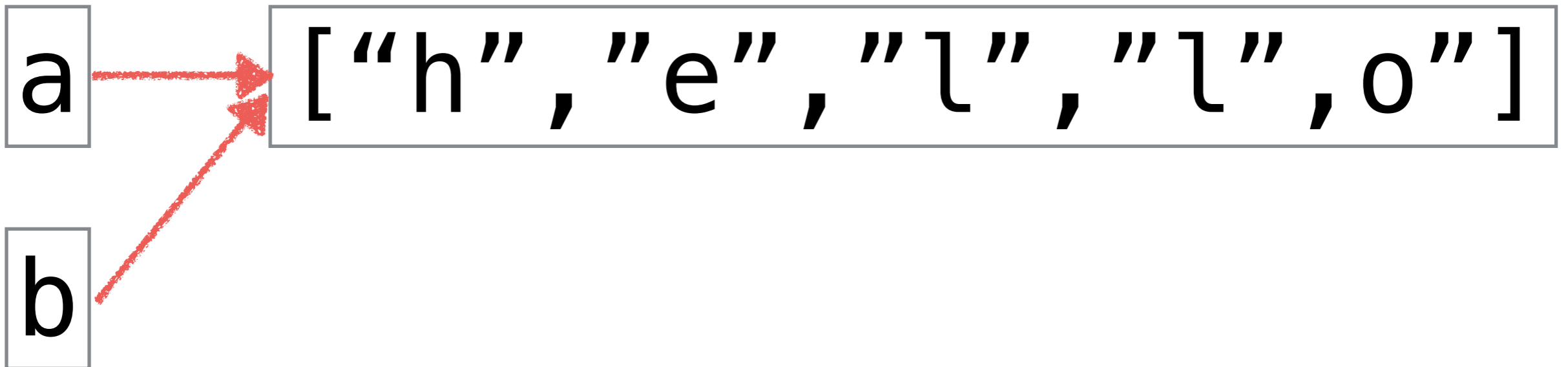
# mutable objects

```
1 a = ["h", "e", "\\", "\\", "o"]  
2 b = ["h", "e", "\\", "\\", "o"]  
3 print(a == b)  
4 print(a is b)
```



# aliasing

```
1 a = ["h", "e", "\t", "\t", "o"]  
2 b = a  
3 print(a == b)  
4 print(a is b)
```



# cloning

```
1 a = ["h", "e", "\t", "\t", "o"]
2 b = a[:]
3 print(a == b)
4 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
- function-modifiers modify the input parameters/arguments
- pure functions communicates with the caller only through parameters
  - do not alter the input parameters
  - create/compute a new data/variable and return reference to it

# 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(anything)`
- creates a list from anything

# 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