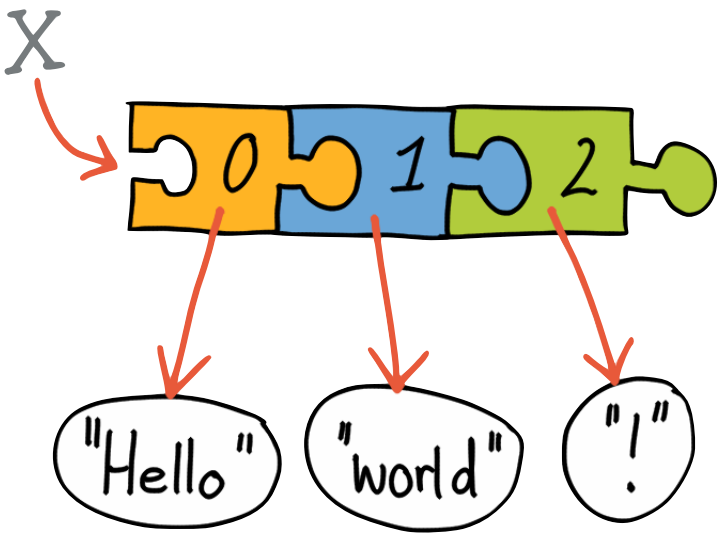
Introduction to Python Programming

08 – Collections: Tuples, Dictionaries

Josef van Genabith (Stefan Thater)
Dept. of Language Science & Technology
Universität des Saarlandes

WS 2022/23

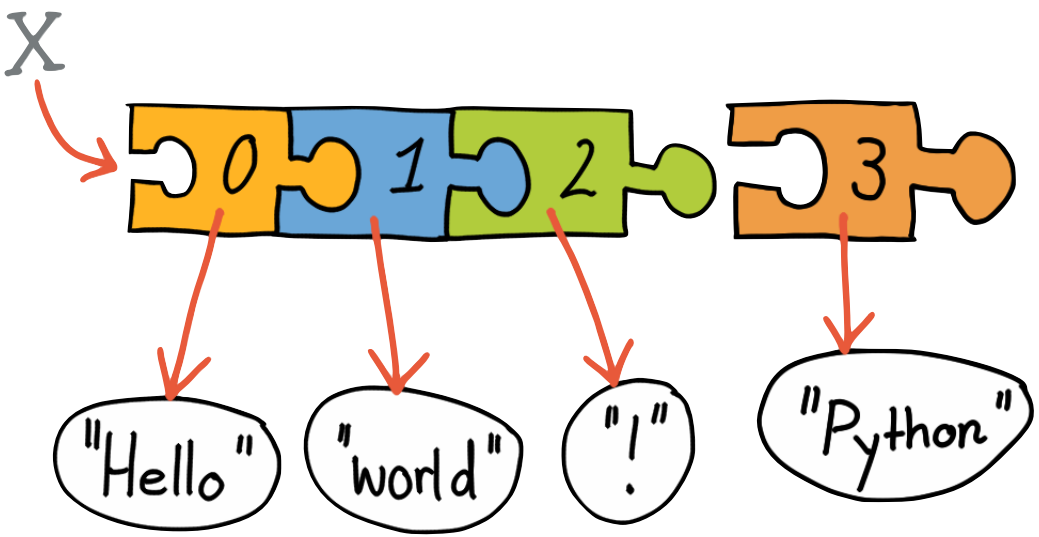
Recap: Lists



Lists are mutable. What does that mean?

```
x = ["Hello", "world", "!"]
```

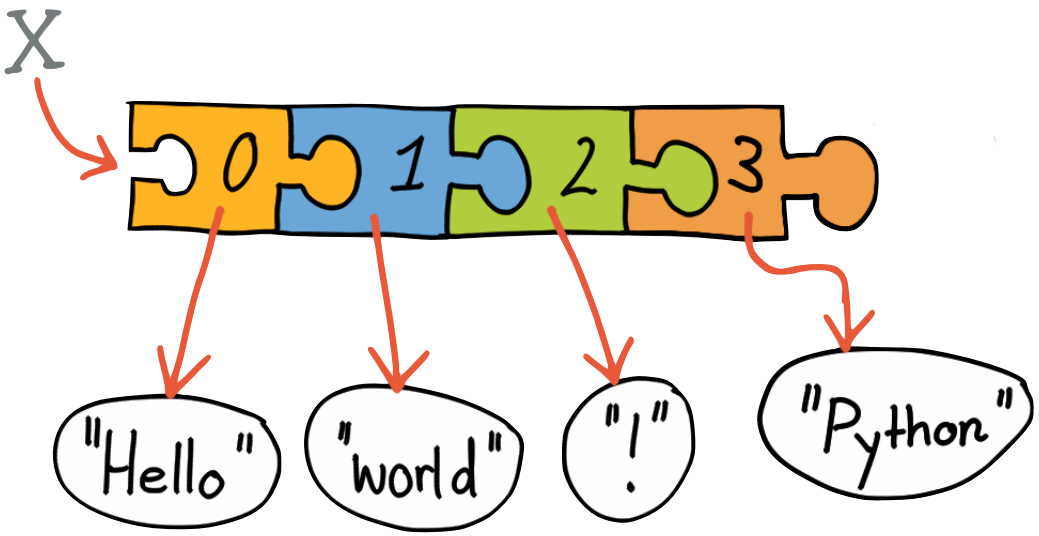
Recap: Lists



How can we add an item to a list?

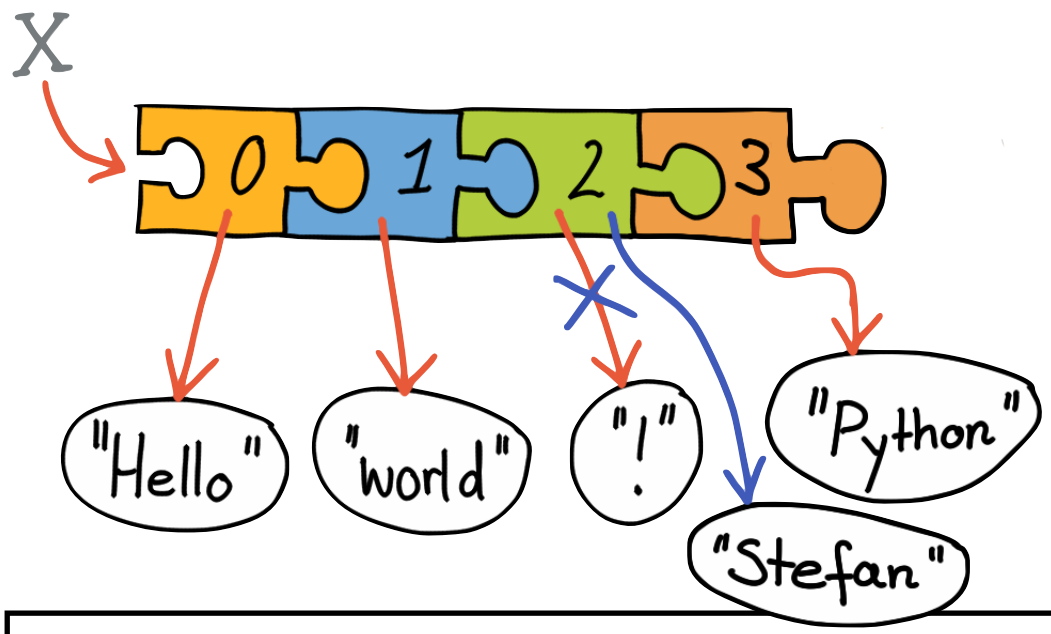
```
x = ["Hello", "world", "!"]
```

Recap: Lists




```
x = ["Hello", "world", "!"]
x.append("Python")
```

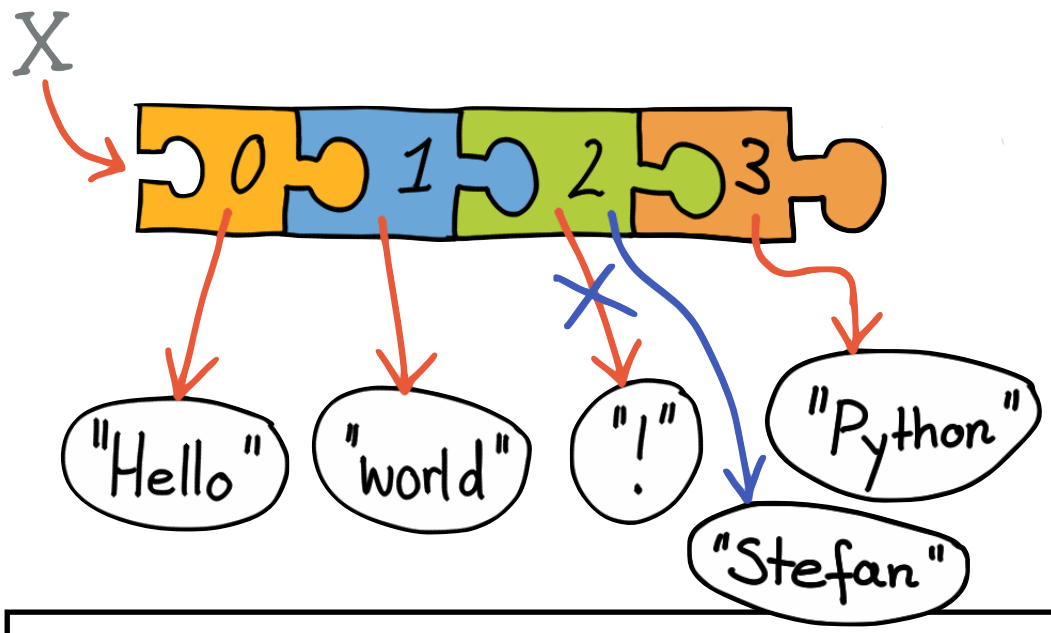
Recap: Lists



How can we replace an item in a list?

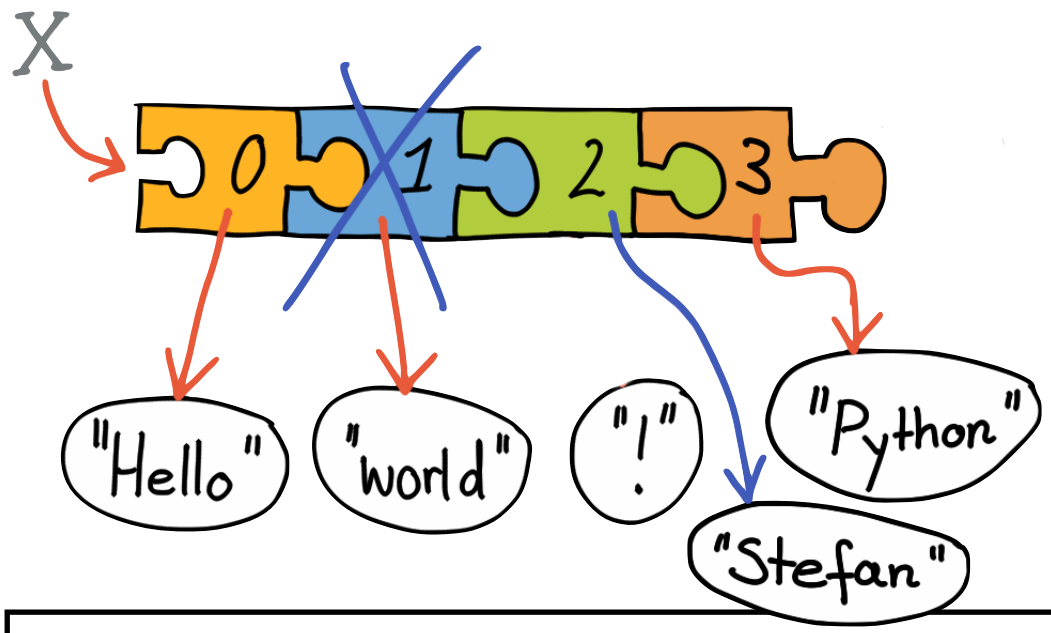
```
x = ["Hello", "world", "!"]
x.append("Python")
```

Recap: Lists




```
x = ["Hello", "world", "!"]
x.append("Python")
x[2] = "Stefan"
```

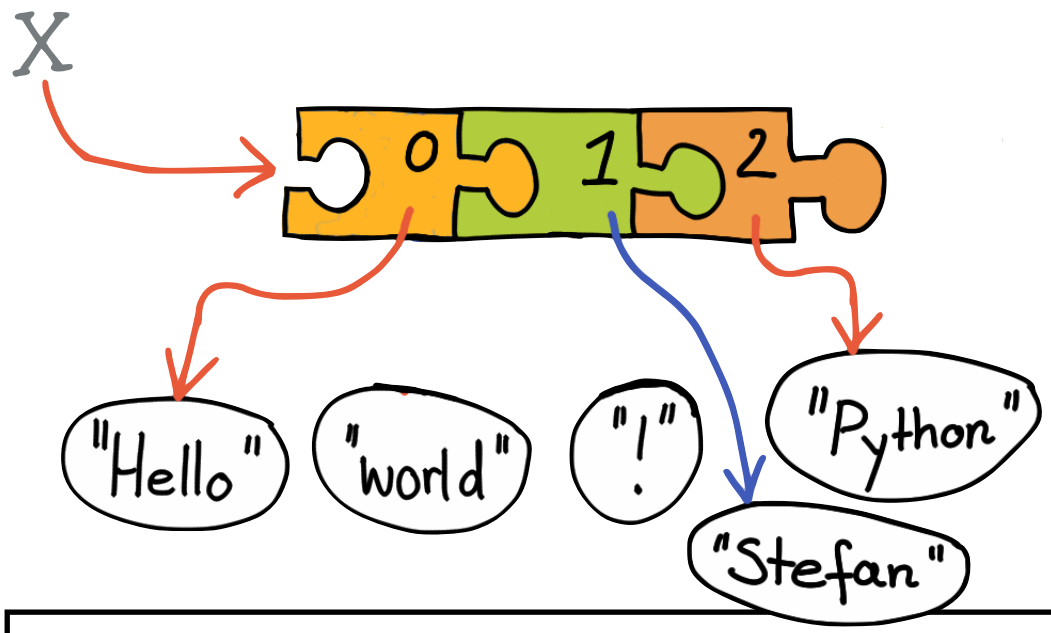
Recap: Lists



How can we delete an item from a list?

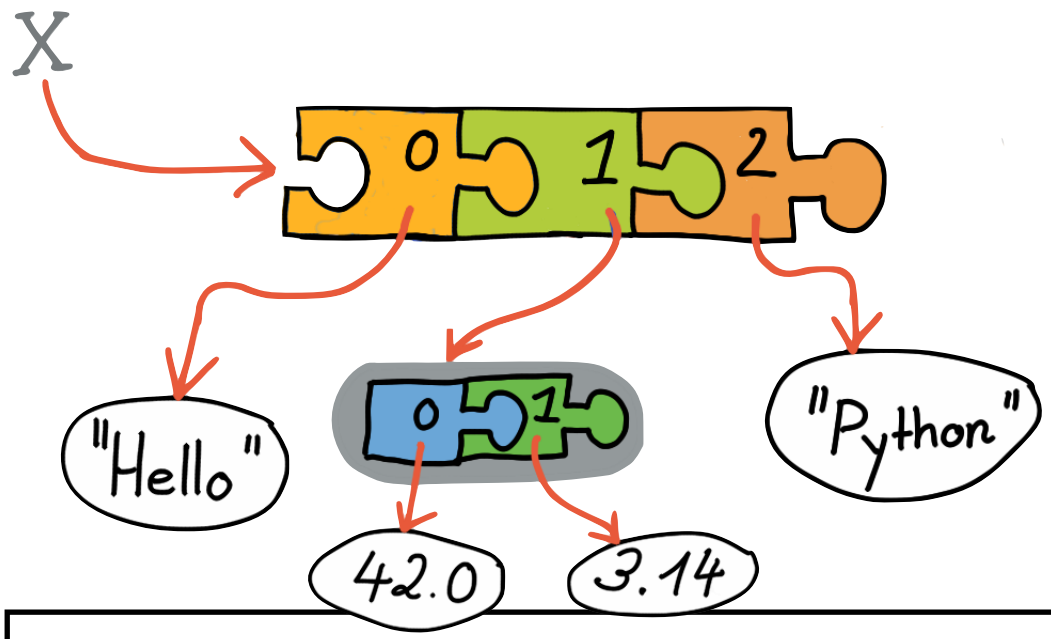
```
x = ["Hello", "world", "!"]
x.append("Python")
x[2] = "Stefan"
```

Recap: Lists




```
x = ["Hello", "world", "!"]
x.append("Python")
x[2] = "Stefan"
del x[1]
```

Recap: Lists

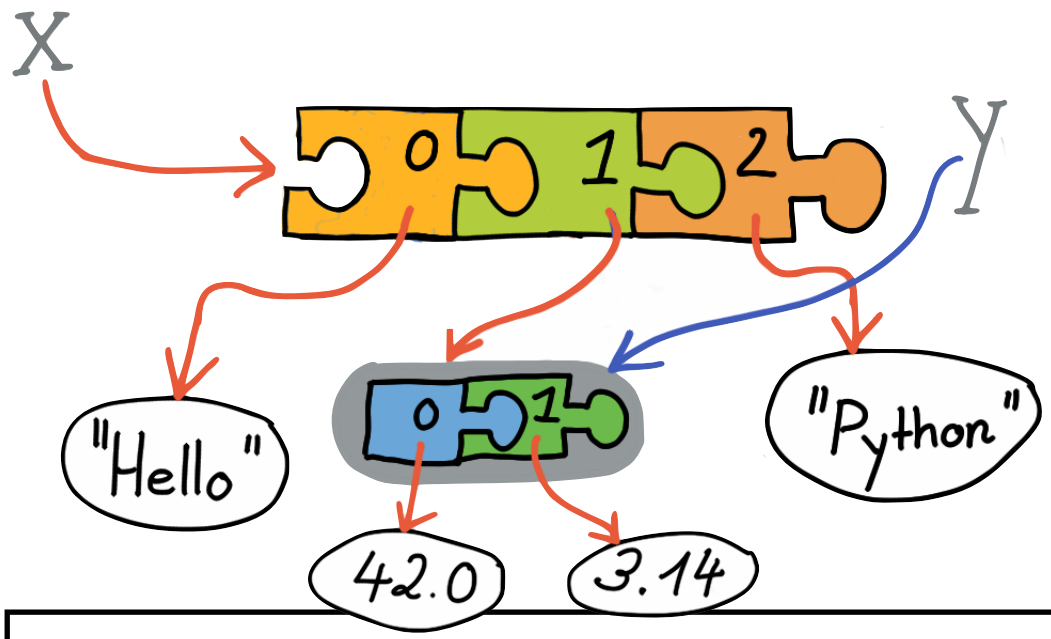


Lists can be nested.

Why do we need to be aware of this?

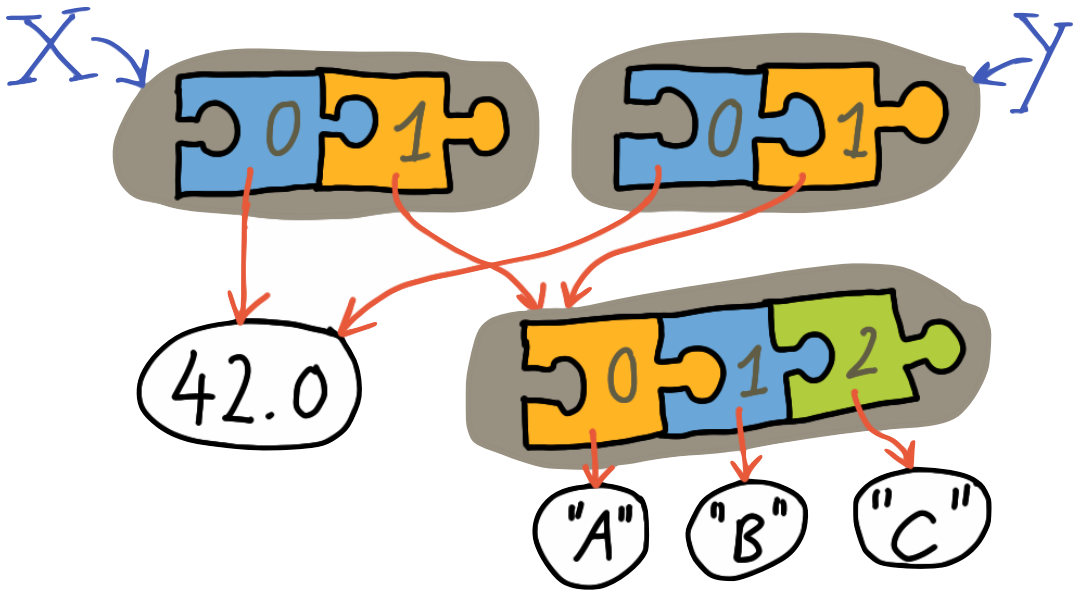
```
x = ["Hello", [42.0, 3.14], "Python"]
```

Recap: Lists



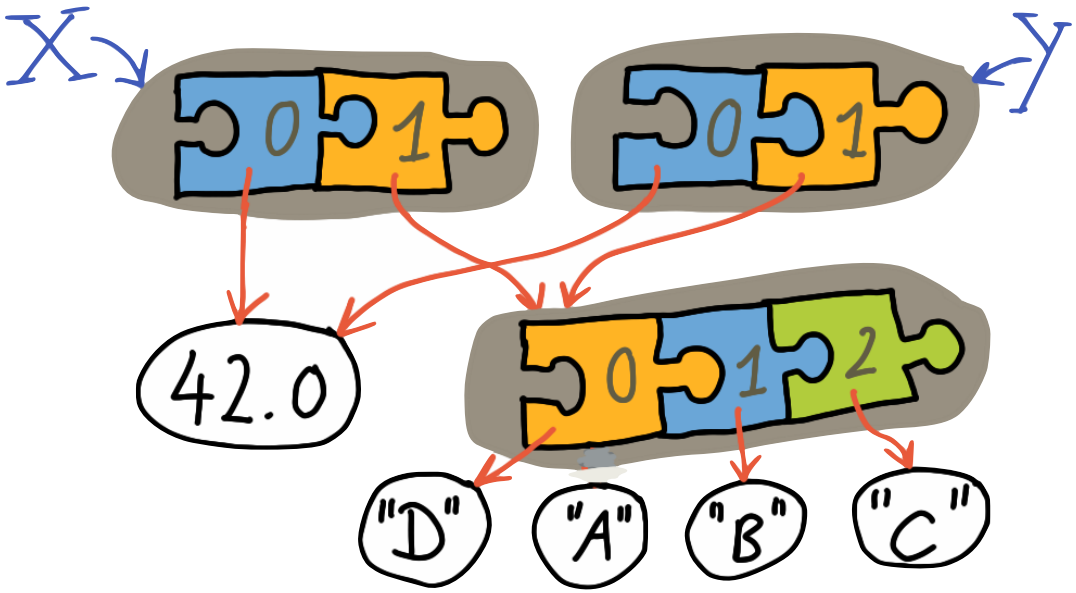

```
x = ["Hello", [42.0, 3.14], "Python"]
y = x[1]
y[1] = 0.0
print(x[1][1])
```

Recap: Slicing



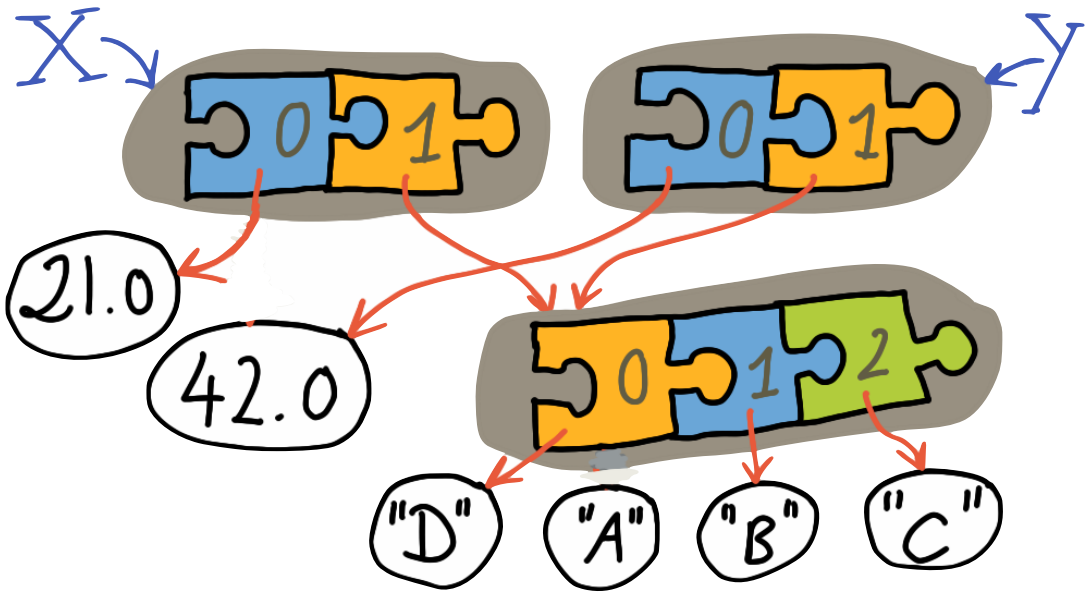

```
x = [42.0, ["A", "B", "C"]]
y = x[:]
```

Recap: Slicing




```
x = [42.0, ["A", "B", "C"]]
y = x[:]
x[1][0] = "D"
print(y[1][0])
```

Recap: Slicing




```
x = [42.0, ["A", "B", "C"]]
y = x[:]
x[0] = 21.0
print(y[0])
```

Recap / Exercise: References

- Create a diagram showing which of the variables point to the same list objects.

```
x = [[1, 2], [3, [5, 6]], 7]
y = x[0][:]
a = x[0]
b = x[1:]
c = x[1][1]
d = b[0][1]
e = b[0]
f = x[1]
```

Tuples

- Tuples are much like lists, except that they are immutable.

```
>>> x = ('a', 'b', 'c')
>>> len(x)
3
>>> x[1]
'b'
>>> x[1:3]
('b', 'c')
>>> x[1:2]
('b',)
>>> x.append('d')
# ERROR
```

Dictionaries

- We often want to store “table-like” information.
- Say, we want to store the telefone numbers of your friends and colleagues:

Key	Value
Smith	3253
Johnson	3938
Brown	1443
Miller	9388

```
myPhoneBook = [("Smith", 3253),
               ("Johnson", 3938),
               ("Brown", 1443),
               ("Miller", 9388)]
```

Exercise

- Implement a function lookup that finds (looks up) the number of a given name.
- (What should be returned in case the name is not found?)

```
myPhoneBook = [("Smith", 3253), ...]

def lookup(phoneBook, name):
    # your code

print(lookup(myPhoneBook, "Brown")) # 1443
```

Solution

```
myPhoneBook = [("Smith", 3253), ..., ("Miller", 9388)]

def lookup(phoneBook, name):
  for (key, value) in phoneBook:
    if name == key:
      return value
  return None # not found

print(lookup(myPhoneBook, "Brown")) # 1443
```

Solution (???)

```
myPhoneBook = [("Smith", 3253), ..., ("Miller", 9388)]

def lookup(phoneBook, name):
  for (key, value) in phoneBook:
    if name == key:
      return value
    else return None # not found

print(lookup(myPhoneBook, "Brown")) # 1443
```

Better Solution: Use `dict`'s

```
myPhoneBook = { "Smith": 3253,
                "Johnson": 3938,
                "Brown": 1443,
                "Miller": 9388 }

print(myPhoneBook["Brown"]) # 1443
```

Dictionaries

- like lists, but indexed by keys (strings, tuples, ...)
- Each key may occur only once per dictionary, i.e. must be unique.
- Keys must be immutable (actually, hashable)
  - strings, tuples, numbers, ...; NOT lists, sets, dicts, ...
- Values can be mutable
- Empty dictionary:
  - `myPhoneBook = dict()`
  - `myPhoneBook = {}`

Dictionaries

- Add a new key-value pair to a dict `d`:
  - `d[key] = value`

```
myPhoneBook = {}
myPhoneBook["Smith"] = 3253
myPhoneBook["Johnson"] = 3938
myPhoneBook["Brown"] = 1443
myPhoneBook["Miller"] = 9388
```

Dictionaries

- Read the value for a key:
  - `d[key]`

```
myPhoneBook = {}
myPhoneBook["Smith"] = 3253
...
...
myPhoneBook["Miller"] = 9388
person = input("Name: ")
print(person, ":", myPhoneBook[person])
```

Dictionaries

- `d[k]` results in an error if `d` has no key `k`.
- You can (and usually should) check for existence of a key using the `in` operator before accessing the dictionary.

```
myPhoneBook = {}
myPhoneBook["Smith"] = 3253
...
...
person = input("Name: ")
if person in myPhoneBook:
  print(person, ":", myPhoneBook[person])
else:
  print("Sorry, I don't know this person.")
```

Dictionaries – Changing entries

```
>>> phoneBook
{'Brown': 1443, 'Smith': 3253, 'Johnson': 3938}
>>> phoneBook['Smith'] = 7777
>>> phoneBook
{'Brown': 1443, 'Smith': 7777, 'Johnson': 3938}
>>> phoneBook["Miller"] = 9388
>>> phoneBook
{'Brown': 1443, 'Smith': 7777, 'Johnson': 3938,
'Miller': 9388}
```

⇒ Assignment of value to key

- If key exists, value is overwritten.
- If key does not exists, new entry is created.

Dictionaries – Comparison of keys

- Keys are internally compared using the == operator.
- Does it return `True`?
- `1` and `1.0` are the same key!

Iteration

`dict.keys()`

- returns a list (actually, a “view”) of all the keys of a dictionary.

`dict.values()`

- returns a view of all the values of a dictionary.

`dict.items()`

- returns a view of all the items in a dictionary. Each item is a tuple: (key, value).

Views are like lists, except we can’t change them.

Iteration

```
phoneBook = {"Brown" : 1443, "Smith" : 3253}
for entry in phoneBook.items():
  print("Name:", entry[0], "Number:", entry[1])

# Alternative:
for (name, number) in phoneBook.items():
  print("Name:", name, "Number:", number)
```

Dictionaries – Deleting Items

```
>>> phoneBook = {'Brown': 1443, 'Smith': 3253,
'Johnson': 3938}
>>> del phoneBook["Smith"]
>>> phoneBook
{'Johnson': 3938, 'Brown': 1443}
>>> number = phoneBook.pop("Johnson")
>>> number
3938
>>> phoneBook
{'Brown': 1443}
```

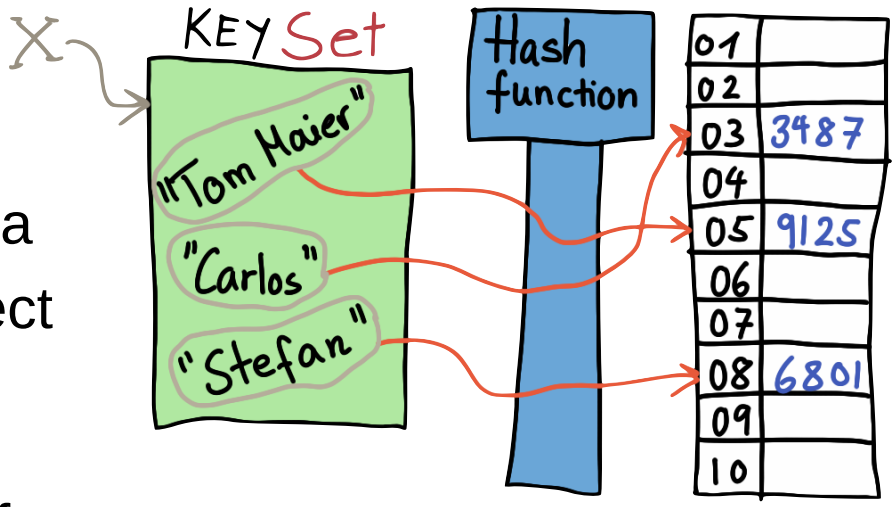
Check whether a key exists before deleting!

- del dict[key] does not return anyting
- dict.pop(key) returns the value

Dictionaries

- like lists, but indexed by keys (strings, tuples, ...)
- Each key may occur only once per dictionary.
- Keys must be immutable (actually, hashable)
  - strings, tuples, numbers, ...; NOT lists, sets, dicts, ...
- Values can be mutable
- Empty dictionary:
  - `myPhoneBook = dict()`
  - `myPhoneBook = {}`

Hashable Objects



- The function `hash(o)` computes a hash value (a number) for an object `o`.
- Dictionaries use the hash-value of the key internally for indexing.
- The hash function is defined only for immutable objects, therefore only immutable objects can be used as keys.

Hashable Objects

```
>>> hash(42)
42
>>> hash("42")
2345936360132995728
>>> hash(True)
1
>>> hash(("John", 4223))
-8594443345497438428
>>>
>>> hash(['A', 'B', 'C'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Summary: Mutability and Hashing

- Immutable = Can not be changed after creation
- Hashable = Can provide a hash value
- All basic types (int, float, bool, str) are both immutable and hashable.
- Collections must be immutable to be hashable.
  - Achtung: Tuples are immutable, but can contain mutable values. Tuples are only hashable if all their content is hashable, too!
- Dictionary keys have to be hashable, while their values can be any kind.

Notes

Notes

Notes

Notes