Python Seminar 2018

Ana Bulović
Jorin Diemer
Jens Hahn

Plan für heute

1. Homework experience
2. Data Types
3. Flow control
4. Kleine Übung

Data Types

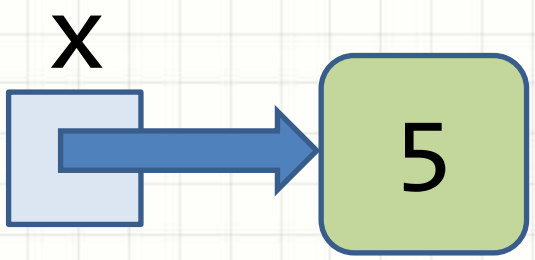
- Numerical and Boolean
  - integers
  - floats
  - Boolean
- Sequence datatypes
  - list
  - set
  - dictionary
  - strings
  - tuples

Mutable

Immutable

Integer

```
x = 5
type(x) is int
```




```
x.bit_length()
```

Mathematical operators

- Addition: +
- Substraction: -
- Multiplication: *
- Division: / or //
- Exponentiate: **
- Modulo: %

Integer methods/attributes

- `bit_length()`
- …

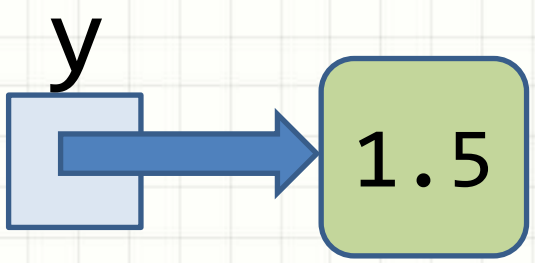
- Check out more: `dir(int)`

Float

```
y = 1.5
type(y) is float
```

Again `dir(float)` lists all float methods and attributes, e.g.

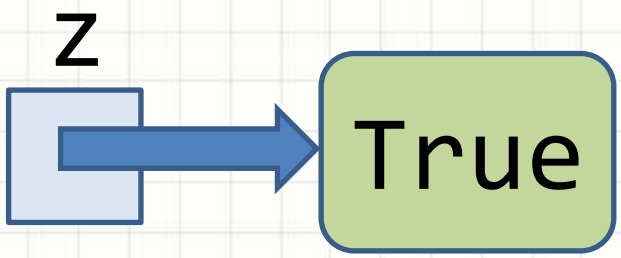
- `as_integer_ratio()`
- `is_integer()`




```
y.is_integer() == False
```

Boolean (True or False)

```
z = True
type(z) is bool
```



Logical Operators

- or/and/not
- |/&

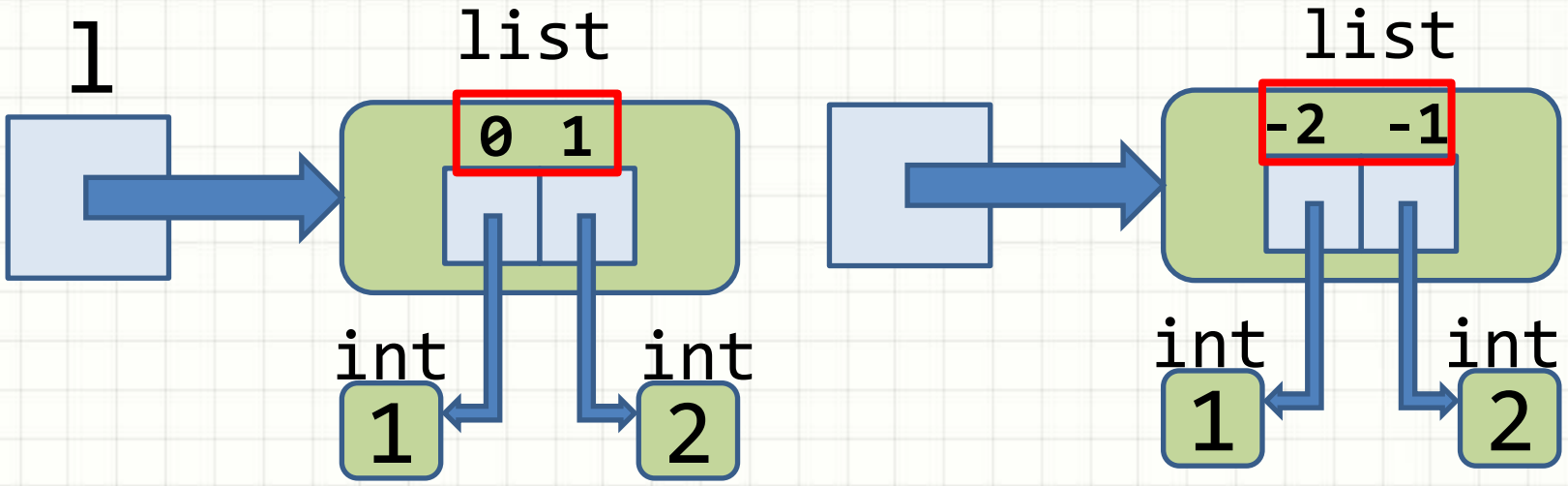
As result of comparisons

Operation	Meaning
<	stricly less than
<=	less tan or equal
>	stricly greater than
>=	greater than or equal
==	equal
!=	not equal
is	obejct identity
is not	negated object identity

List

```
l = [1, 2]
type(l) is list
```

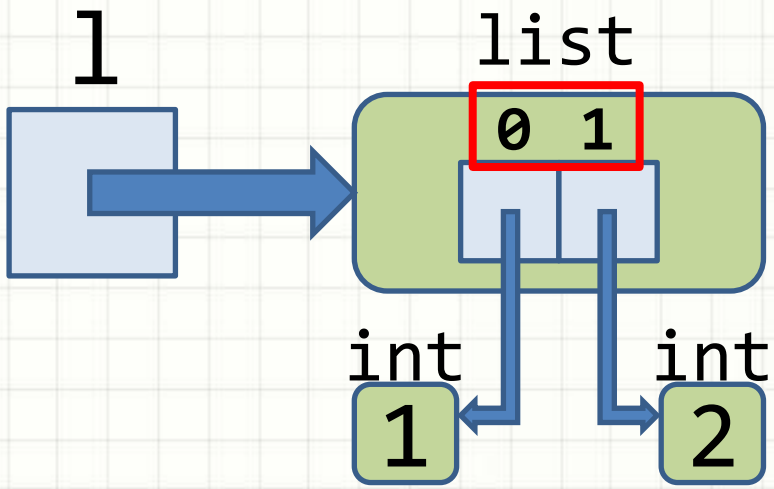
- Ordered sequence of references
- References have **two** indices



List indexing

```
l = [1, 2]
type(l) is list
```

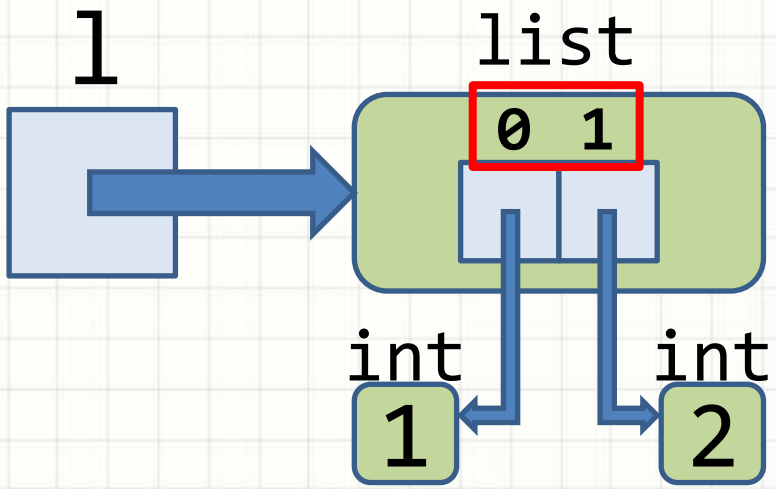
```
l[0] == 1
l[1] == 2
l[-2] ==
```



List indexing

```
l = [1, 2]
type(l) is list
```

```
l[0] == 1
l[1] == 2
l[-2] == 1
```



List slicing

```
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
l[1:4] == [2, 3, 4]
l[2:7:2] == [3, 5, 7]
l[::-1] == [9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Pattern:
list_object[start:stop:step]

- stop is exclusive!!!

List methods

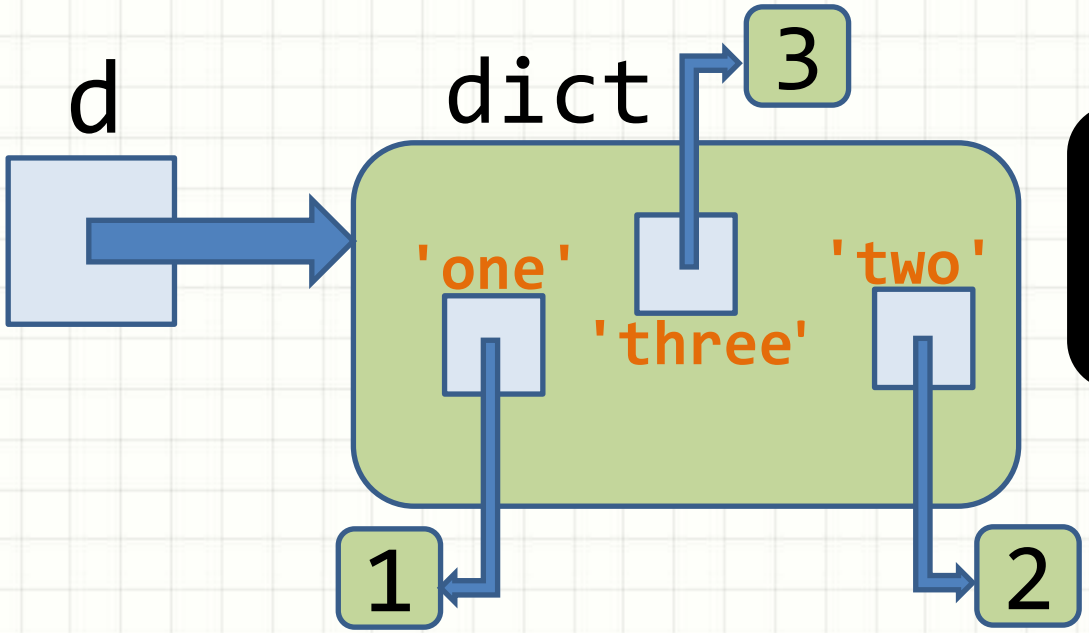
```
l = [1, 2]
```

Method	Current list	Return value
l.append(3)	[1, 2, 3]	None
l.append([4, 5])	[1, 2, 3, [4, 5]]	None
l.extend([3, 4, 5])	[1, 2, 3, [4, 5], 3, 4, 5]	None
l.pop(3)	[1, 2, 3, 3, 4, 5]	[4, 5]
l.reverse()	[5, 4, 3, 3, 2, 1]	None
l.sort()	[1, 2, 3, 3, 4, 5]	None
l.count(3)	[1, 2, 3, 3, 4, 5]	2
l.index(3)	[1, 2, 3, 3, 4, 5]	2
l.insert(1, 6)	[1, 6, 2, 3, 3, 4, 5]	None
l.remove(4)	[1, 6, 2, 3, 3, 5]	None
l.clear()	[]	None

Dictionary

```
d = {'one': 1,
     'two': 2,
     'three': 3}
type(d) is dict
```

- unordered collection of **key**:**value** pairs
- **keys** have to be immutable




```
d['one'] == 1
d['four'] = 4
```

Dictionary methods

- `d.keys()`: returns list of keys
- `d.values()`: returns list of values
- `d.get(key, default)`: returns value of key or default if key is not in *d*
- `'key1' in d`: to check if a key is in the dictionary d
- use `dir(dict)` to see them all

Set

```
s = {'one', 'two', 'three'}
s = set('one', 'two', 'three')
type(s) is set
```

- unordered collection with no duplicates
- logical and set operations

```
s1 = set([1, 2, 1, 3, 2, 3, 4])
s2 = {1, 2, 3, 4}
s1 == s2
```

Set – logical operators

```
s1 = {'a', 'b', 'c'}
s2 = {'c', 'd', 'e'}
s1 == s2
```

Operation		Return value
`s1 - s2`		`{'a', 'b'}`
`s1|s2`	Or	`{'a', 'b', 'c', 'd', 'e'}`
`s1&s2`	And	`{'c'}`
`s1^s2`	Xor	`{'a', 'b', 'd', 'e'}`

Tuple

```
t = 'one', 'two', 3
t = ('one', 'two', 3)
t = tuple(['one', 'two', 3])
type(t) is tuple
```

- ordered immutable collection of references
- useful to pass parameters of different type
- tuple unpacking

```
eins, zwei, drei = t
```

String

```
st = 'meep'
st = ''meep''
st = '''meep'''
type(st) is str
```

- immutable ordered sequence of unicode points
- triple quotes allow multi-line strings
- can be sliced and indexed as a list

- st.capitalize()
- st.endswith()
- st.startswith()
- st.upper()
- st.lower()
- …

For loop

```
for x in iterable:
. . . . print(x)
```

- iterable has to be sequence type (`list, string, tuple, dict, set`)
- x is the newly defined running variable
- indented code (**statement**) is executed for each iteration step

```
count = 0
for x in [1, 2, 3]:
      count += x
count == 6
```

If/elif/else

```
if n==2:
      print(n)
elif n==3:
      print(n - 1)
else:
      print(2)
```

- **expression** after `if` is evaluated
- If **expression** is True **statement** is executed
- `elif`(else if) and `else` are optional
- **expression** after `elif` is evaluated if expressions before are `False`
- You can have multiple `elif`‘s
- The code after `else` is executed if all preceding expressions are False

while loop

```
x = 1
while x < 5:
    x +=1
```

```
x = 1
while x < 5:
    x -=1
```

- `while` iterates over block of indented code as long as the **expression** is True
- it is used when the number of iteration steps is not known before
- be careful not to end up in infinite loop
- if possible use `for` loop

HAUSAUFGABEN