UNIVERSITÄT BIELEFELD
Faculty of Technology

Programming

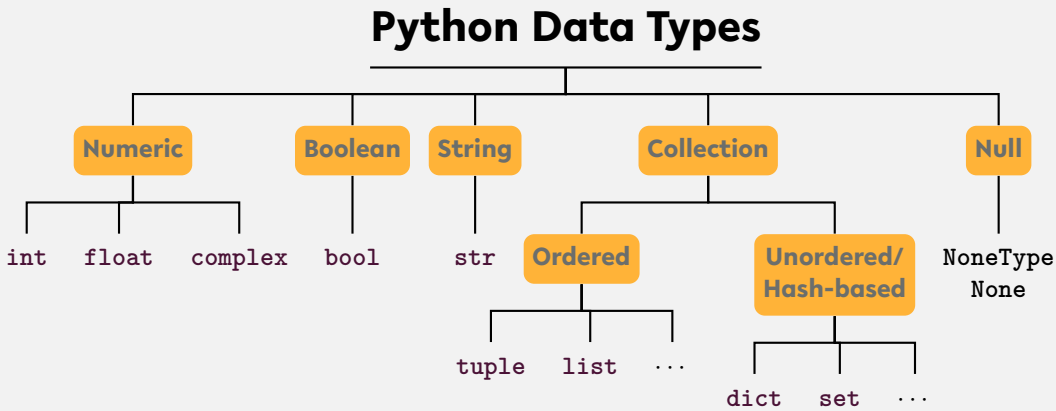
Programming & Python Basics

Harsha Manjunath

Faculty of Technology,

Bielefeld University

Recap



... and user-defined types

String

`str()`

- instantiation: `s = 'a new string'` or `s = "a new string"`
- length: `len(s)`
- access:
  - first: `s[0]`
  - slice: `s[1:3]`
  - last: `s[-1]`
- existence: `'n' in s` or `'new' in s`
- frequency: `s.count('new')`

UNIVERSITÄT BIELEFELD

List

`list()`

- instantiation: `l = [1, 2, 3]`
- length: `len(l)`
- add elements: `l.append("content")`
- access:
  - first: `l[0]`
  - slice: `l[1:3]`
  - last: `l[-1]`
- existence: `2 in l`
- location: `l.index(3)`

Complex data: Mappings

`dict()`

- instantiation: `d = dict()`, `d = {'x': 1, 'y': 2 }`, …
- length: `len(l)`
- add elements: `d['a'] = 'ef'`
- access: `d['a']`
- existence: `'a' in d`

Programming Basics

Data Types & Mutability

Evaluation Order

Conditions & Comparisons

Conditional statements: `if/else` clause

```
if «Boolean expression»:
␣␣␣␣«statement»
```

⚠ Mind the indention!

OR

```
if «Boolean expression»:
␣␣␣␣«statement»
else:
␣␣␣␣«alternative statement»
```

Conditional statements: if/else

```
1 a = True
2 if a:
3     print('a is True')
4
5     if 'this is a text':
6         print('another true statement')
```

Conditional statements: `if/else`

```
1 a = True
2 if a:
3     print('a is True')
4 else:
5     print('a is False')
```

Boolean operators and comparisons

Elementary logic: `and`, `or`, `not`

Variables		Boolean expression		
`a`	`b`	`not a`	`a and b`	`a or b`
`False`	`False`	`True`	`False`	`False`
`False`	`True`	`True`	`False`	`True`
`True`	`False`	`False`	`False`	`True`
`True`	`True`	`False`	`True`	`True`

Comparisons: Operators

- `==` “is equal/equivalent to”
- `!=` “is not equal/equivalent to”
- `>` “is larger than”
- `<` “is is smaller than”
- `>=` “is larger or equal to”
- `<=` “is smaller or equal to”
- `is` “is identical instance of”
- `is not` “is not identical instance of”
- `in` “is contained in collection”
- `not in` “is not contained in collection”

Conditional execution based on comparisons

```
1 a = 4.0
2 b = 2.0
3 if not a > b:
4     print('true statement!')
```

Conditional execution based on comparisons

```
1 a = 'this is a text'
2 b = 'this is a text'
3 if a >= b:
4     print('true statement!')
```

Conditional execution based on comparisons

```
1 a = 'this is a text'
2 if a:
3     print('true statement!')
```

Conditional execution based on comparisons

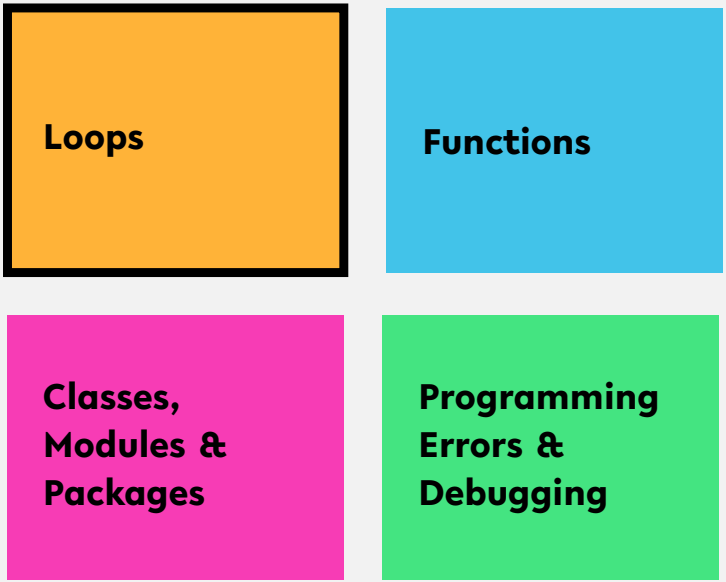
```
1 a = list()
2 b = list()
3 if a is b:
4     print('true statement!')
```

Conditional execution based on comparisons

```
1 a = list()
2 b = list()
3 if a == b:
4     print('true statement!')
```

Conditional execution based on comparisons

```
1 a = list()
2 b = 1
3 if b not in a:
4     print('b is contained in collection a')
5 else:
6     print('b is not contained in collection a')
```


Loops
Functions
Classes, Modules & Packages
Programming Errors & Debugging

`for`-Loop

`for` «*control variable name*» `in` «*iterable*»:
␣␣␣␣«*statement*»

⚠ Mind the indention!

`for`-Loop: Iteration over ordered collections

Loop over elements

```
1 # tuple filled with arbitrary elements
2 my_tuple = (1, 2.0, 'text', list(), dict())
3
4 # for-loop over my_tuple with control
    variable 'el'
5 for el in my_tuple:
6     msg = 'element: {}'.format(el)
7     print(msg)
```

`for`-Loop: Iteration over ordered collections

Loop over indices with `range`

```
1  # tuple filled with arbitrary elements
2  my_tuple = (1, 2.0, 'text', list(), dict())
3
4  # for-loop over my_tuple with control
     variable 'i'
5  for i in range(len(my_tuple)):
6      el = my_tuple[i]
7      msg = 'element {}: {}'.format(i, el)
8      print(msg)
```

for-Loop: Iteration over ordered collections

Update `list` in for-loop

```
1  # list filled with arbitrary elements
2  my_list = [1, 2.0, 'text', list(), dict()]
3
4  # for-loop over my_list with control
     variable 'i'
5  for i in range(len(my_list)):
6      # update element with index i
7      my_list[i] = 'element {}: {}'.format(i,
         my_list[i])
8      print(my_list[i])
```

`for`-Loop: Iteration over ordered collections

Loop over indices and elements with `enumerate`

```
1  # list filled with arbitrary elements
2  my_list = [1, 2.0, 'text', list(), dict()]
3
4  # for-loop over my_list with control
     variables 'i' and 'el'
5  for i, el in enumerate(my_list):
6      # update element with index i
7      my_list[i] = 'element {}: {}'.format(i,
         el)
8      print('old: {}, new: {}'.format(el,
         my_list[i]))
```

`for`-Loop: Iteration over unordered collections

Loop over elements of a `set`

```
# set filled with arbitrary elements
my_set = {1, 1, 1, 2.0, 'text'}

# for-loop over my_set with control variable 'el'
for el in my_set:
    msg = 'element: {}'.format(el)
    print(msg)
```

`for`-Loop: Iteration over unordered collections

Loop over keys of a `dict`

```
1 # dictionary filled with arbitrary elements
2 my_dict = {'key': 'value', 1: 'text', (1, 2)
    : 'text'}
3
4 # for-loop over keys of my_dict with control
    variable 'key'
5 for key in my_dict:
6     val = my_dict[key]
7     msg = 'key: {}, value: {}'.format(key,
        val)
8     print(msg)
```

`for`-Loop: Iteration over unordered collections

Loop over items of a `dict`

```
1 # dictionary filled with arbitrary elements
2 my_dict = {'key': 'value', 1: 'text', (1, 2)
    : 'text'}
3
4 # for-loop over items of my_dict with
    control variables 'key', 'val'
5 for key, val in my_dict.items():
6     msg = 'key: {}, value: {}'.format(key,
        val)
7     print(msg)
```

Conditional iteration

Another type of loop in Python: `while`

- Loops until condition becomes True

```
1 x = 5
2 while x > 0:
3     print(x)
4     x -= 1 # shorthand for x = x - 1
```

Special keywords in loops:

- `continue`: aborts current iteration and continues with the next
- `break`: aborts loop completely

Quiz

- What does the instruction `tuple(range(3))` return?

  `[1, 2, 3]` `(1, 2, 3)` `(0, 1, 2)` `(0, 1, 2, 3)`

- Let *x* be any integer, how many times is the `print` statement in the follwing `for`-loop executed?

```
1 for i in range(x):
2     for j in range(i):
3         print((i, j))
```

Quiz

- What does the instruction `tuple(range(3))` return?

  `[1, 2, 3]` `(1, 2, 3)` `(0, 1, 2)` ✔ `(0, 1, 2, 3)`

- Let x be any integer, how many times is the `print` statement in the follwing `for`-loop executed?

```
1 for i in range(x):
2     for j in range(i):
3         print((i, j))
```

$\binom{x}{2}$ times