

Programming

Loops

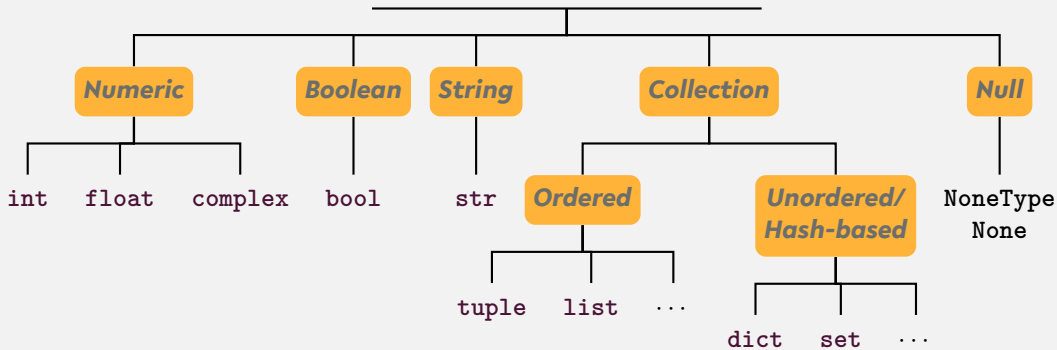
Luna Pianesi

Faculty of Technology, Bielefeld University

```
332
333
334     extrapolate = None:
335     x = np.asarray(x)
336     x_shape, x_ndim = x.shape, x.ndim
337     x = np.ascontiguousarray(x.ravel(), dtype=np
338
339     # With periodic extrapolation we map x to the
340     # [self.t[k], self.t[n]].
341     if extrapolate == 'periodic':
342         n = self.t.size - self.k - 1
343         x = self.t[self.k] + (x - self.t[self.k]) *
344         extrapolate = False
345
346     out = np.empty((len(x), prod(self.c.shape[1:])),
347 self._ensure_c_contiguous()
348 self._evaluate(x, nu, extrapolate, out)
349 out = out.reshape(x_shape + self.c.shape[1:])
350 if self.axis != 0:
351     # transpose to move the calculated values to t
352     l = list(range(out.ndim))
353     l = l[x_ndim:x_ndim+self.axis] + l[:x_ndim] +
354     out = out.transpose(l)
355 return out
356 def _evaluate(self, xp, nu, extrapolate, out):
357     _bspl.evaluate_spline(self.t, self.c.reshape(self.c
358 self.k, xp, nu, extrapolate, out)
359
360 def _ensure_c_contiguous(self):
361     """
362     c and t may be modified by the user. The Cython code
363     c and t may be C contiguous.
364     """
365     if not (self.c.flags.c_contiguous and
366            self.t.flags.c_contiguous):
367         self.c = np.ascontiguousarray(self.c)
368         self.t = np.ascontiguousarray(self.t)
```

Recap

Python Data Types



... and user-defined types

Conditional statements: if/else clause

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

 **Mind the indentation!**

OR

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

Boolean operators, Comparisons

- ❖ Elementary logic: `and`, `or`, `not`
- ❖ Comparators:
 - ❖ `==` “is equal/equivalent to”
 - ❖ `!=` “is not equal/equivalent to”
 - ❖ `>` “is larger than”
 - ❖ `<` “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”

Loops

What are loops?

- ❖ Loops are the ability of programming languages to execute something again and again
- ❖ They are a **control flow statement**
- ❖ They allow us to execute a group of instructions as long as the initial condition remains satisfied
- ❖ Two *keywords*: **for** and **while**

For loops

While loops

for-Loop

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

⚠ Mind the indentation!

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**

```
1 # set filled with arbitrary elements
2 my_set = {1, 1, 1, 2.0, 'text'}
3
4 # for-loop over my_set with control variable
   'el'
5 for el in my_set:
6     msg = 'element: {}'.format(el)
7     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)
3           : 'text'}
4
5 # for-loop over keys of my_dict with control
6 variable 'key'
7 for key in my_dict:
8     val = my_dict[key]
9     msg = 'key: {}, value: {}'.format(key,
10    val)
11     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)
3           : 'text'}
4
5 # for-loop over items of my_dict with
6   control variables 'key', 'val'
7 for key, val in my_dict.items():
8     msg = 'key: {}, value: {}'.format(key,
9                                       val)
10    print(msg)
```

For loops

While loops

Conditional iteration

Another type of loop in Python: `while`

- Loops until condition becomes False

```
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 following `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 following `for`-loop executed?

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

$\binom{x}{2}$ times

Recap

Summary

➤ `for` and `while`