Rail Data Science

Prof. Dr. Raphael Pfaff
Lehrgebiet Schienenfahrzeugtechnik

May 23, 2023

1 Basic data types and structures in Python

1.1 Typing:

Python is a dynamically typed language, i.e. there is no declaration of variable type required.

[1]:

```
x = 'five'
print(x +': '+ str(type(x)))
x = 5
print(str(x)+': '+ str(type(x)))
```

```
five: <class 'str'>
5: <class 'int'>
```

1.2 Numbers:

Python (without additional packages) supports:

- Integers: `x = 1`
- Floats: `x = 1.0`or `x=1e-12`
- Complex numbers: `z=1+2j`

[2]:

```
# Define some numbers and print them
x = 1
print(x)
y = 1.5
print(y)
z = 1+2j
print(z)

# Calculate a bit:
print('Result is in superclass, x+y=')
print(x+y)
print('Result is in superclass, x+y+z=')
print(x+y+z)
```

```
1
1.5
(1+2j)
```

```
Result is in superclass, x+y=
2.5
Result is in superclass, x+y+z=
(3.5+2j)
```

1.2.1 Type conversions:

It is possible to convert between the number types by explicit operations:

- Integer to float
- Float to integer
- Float or integer to complex

[3]:
```
print(float(x))
print(int(y))
print(complex(x))
print(complex(y))
```

```
1.0
1
(1+0j)
(1.5+0j)
```

1.2.2 Boolean:

Can take `True`or `False`, however can be use similar to numbers 1 and 0:

[4]:
```
A = True
B = False
11*A + 7*B
```

[4]:
```
11
```

1.3 Strings:

Strings are arrays of letters, so single parts of texts can be addressed. Declare by either `"string"`or `'string'`.

[5]:
```
text = "John"
print(text)
print(text[1:3]) # Strings are list of letters, here second and third are␣
 ↪selected
print(len(text)) # How long is John's name?
```

```
John
oh
4
```

Concatenate two strings by using `string1 + string2`

[6]:

```
string1 = 'Hello ' #Observe the space
string2 = 'World!'
string12 = string1 + string2
print(string12)
```

```
Hello World!
```

Convert number to string (e.g. for printing):

[7]:

```
complexNumber = str(z)
print('Nice complex number: ' + complexNumber)
```

```
Nice complex number: (1+2j)
```

1.4 Data structures

1.4.1 Lists

Lists can be changed after instantiation, i.e. they are considered mutable. Lists may contain any combination of data types (objects in Python therminology).

They are constructed using square brackets `[obj1, ..., objn]`.

[8]:

```
l = ["apple", "banana", "cherry", 1]
print(l)
```

```
['apple', 'banana', 'cherry', 1]
```

Single items can be addressed in a MATLAB-style syntax, with negative entries being addressed from the end of the list as well as a colon operator (Attention, starts at 0 and ends **before** n-th element!):

[9]:

```
print(l[1]) # Print only the second item
print(l[-1]) # Final element
print(l[0:3]) # First three elements
```

```
banana
1
['apple', 'banana', 'cherry']
```

Using `.append`, it is possible to add an item to the end of the list, very useful in loops.

[10]:

```
l.append('pear') # Append an item
print(l)
```

```
['apple', 'banana', 'cherry', 1, 'pear']
```

List items can be changed:

[11]:

```
l[3] = 'orange' # Change item 4
print(l)
```

```
['apple', 'banana', 'cherry', 'orange', 'pear']
```

Useful list functionalities Loop through lists

[12]:

```
for x in l:
    print(x)
```

```
apple
banana
cherry
orange
pear
```

Conditional statement based on list

[13]:

```
if "apple" in l:
    print("Yes, 'apple' is in the fruits list")
```

```
Yes, 'apple' is in the fruits list
```

List length

[14]:

```
len(l)
```

[14]:

```
5
```

Create an empty list

[15]:

```
l2 = []
```

1.4.2 Tuples

Tuples are ordered collections of objects and cannot be changed after instantiation.

They are constructed using brackets `(obj1, ..., objn)`.

[16]:

```
t = ("apple", "banana", "cherry")
print(t)
print(t[1]) # Print only the second item
print(t[1:3]) # Range of elements
print(t[-1]) # Final element
```

```
('apple', 'banana', 'cherry')
banana
('banana', 'cherry')
cherry
```

Tuple functionality Similar to lists, it is possible to loop through tuples, to inspect for truth value and to obtain the length of a tuple:

[17]:
```
for x in t:
    print(x)

if 'apple' in t:
    print('There is apple in the tuple!')

len(t)
```

```
apple
banana
cherry
There is apple in the tuple!
```

[17]:
```
3
```

1.4.3 Sets

Sets are unordered data structures that can be changed after instantiation. They support certain set operations such as `union()` and `intersection()`.

They are constructed using curly brackets `{obj1, ..., objn}`.

[18]:
```
s = {"apple", "banana", "cherry"}
s2 = {"banana", "cherry", "orange"}
```

Set operations:

- $S \cap S_2$: `s.intersection(s2)`
- $S \cup S_2$: `s.union(s2)`

Both return a set.

[19]:
```
print(s.intersection(s2))
print(s.union(s2))
```

```
{'cherry', 'banana'}
{'orange', 'banana', 'apple', 'cherry'}
```

Loop through sets:

[20]:
```
for x in s.intersection(s2):
    print(x)
```

```
cherry
banana
```

Add and remove items:

[21]:
```
s.add('pear')
print(s)
```

```
{'apple', 'pear', 'cherry', 'banana'}
```

```
[22]: s.remove('banana')
      print(s)
```

```
{'apple', 'pear', 'cherry'}
```

1.4.4 Dictionaries (Dicts)

In addition to lists, dicts are perhaps the most dominant data structure in Python, as they can handle self explanatory key-value pairs.

They are constructed using curly brackets and `'key': value`pairs separated by commas.

```
[23]: d ={
        "make": "Bombardier",
        "model": "Traxx",
        "power": 5600
      }
      print(d)
```

```
{'make': 'Bombardier', 'model': 'Traxx', 'power': 5600}
```

Access model:

```
[24]: d['model']
```

```
[24]: 'Traxx'
```

Change power:

```
[25]: d['power'] = 4200
```

Add key value pair:

```
[26]: d['year'] = 2016
      print(d)
```

```
{'make': 'Bombardier', 'model': 'Traxx', 'power': 4200, 'year': 2016}
```

1.5 Exercise

1. Use `range(0,n)`to generate a linear list of integers, let $n = 10$
2. Loop through this list
3. For each integer i, calculate and print (nicely!)

- i squared: i^2
- Factorial of i: $i! = i(i-1)!$, $0! = 1$
- Append both values to list that you initialise empty

```
[ ]:
```