Programming

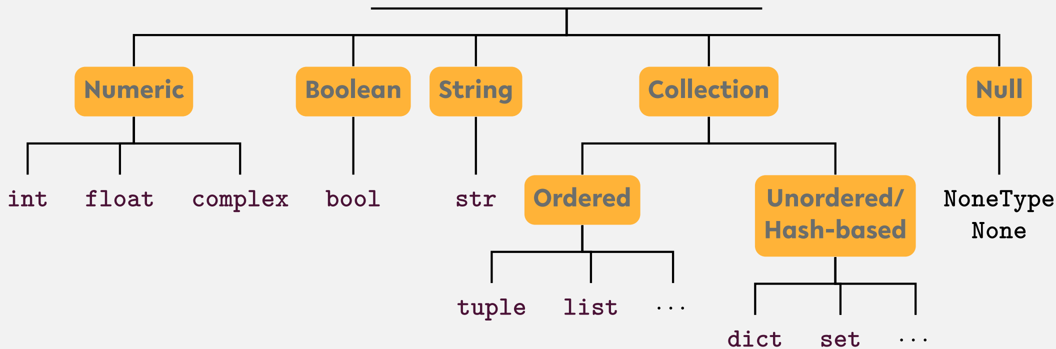
Programming & Python Basics

Daniel Dörr

Faculty of Technology, Bielefeld University

Recap

Python Data Types



... 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')`

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`

Conditional statements: `if/else` clause

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

Mind the indention!

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 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

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

```python
# tuple filled with arbitrary elements
my_tuple = (1, 2.0, 'text', list(), dict())

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

for-Loop: Iteration over ordered collections

Loop over indices with range

```
# tuple filled with arbitrary elements
my_tuple = (1, 2.0, 'text', list(), dict())

# for-loop over my_tuple with control
   variable 'i'
for i in range(len(my_tuple)):
    el = my_tuple[i]
    msg = 'element {}: {}'.format(i, el)
    print(msg)
```

for-Loop: Iteration over ordered collections

Update list in for-loop

```
# list filled with arbitrary elements
my_list = [1, 2.0, 'text', list(), dict()]

# for-loop over my_list with control
    variable 'i'
for i in range(len(my_list)):
    # update element with index i
    my_list[i] = 'element {}: {}'.format(i,
        my_list[i])
    print(my_list[i])
```

for-Loop: Iteration over ordered collections

Loop over indices and elements with enumerate

```
# list filled with arbitrary elements
my_list = [1, 2.0, 'text', list(), dict()]

# for-loop over my_list with control variables 'i' and 'el'
for i, el in enumerate(my_list):
    # update element with index i
    my_list[i] = 'element {}: {}'.format(i, el)
    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`

```
# dictionary filled with arbitrary elements
my_dict = {'key': 'value', 1: 'text', (1, 2)
   : 'text'}

# for-loop over keys of my_dict with control
    variable 'key'
for key in my_dict:
    val = my_dict[key]
    msg = 'key: {}, value: {}'.format(key,
       val)
    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

```
x = 5
while x > 0:
    print(x)
    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?

```
for i in range(x):
    for j in range(i):
        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?

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

$\binom{x}{2}$ times

Loops

Functions

Classes, Modules & Packages

Programming Errors & Debugging

Functions

```
def «functionName» ( «parameterName1», «parameterName2», ...  ):
␣␣␣␣«statement»
␣␣␣␣return «statement»
```

⚠ Mind the indention!

gray = optional

Variable Scope

Functions have a separate variable scope!

- internal variables are not accessible from outside
- calling global functions and variables is possible
  - Reading global variables is discouraged
- Changing global variables requires

  «`global variableName`»

  statement inside function (highly discouraged)

source: https://www.learnpython.org/en/Functions

Functions—a simple example

```
1 def myFirstFunction():
2     print('this is my first function')
3
4 # call function
5 myFirstFunction()
6
7 # save return value in variable
8 hereComesNothing = myFirstFunction()
```

Functions—example of code reuse

```
1  def findSubstringInStrings(stringCollection, pattern):
2      occ = list()
3      for i, s in enumerate(stringCollection):
4          j = s.find(pattern)
5          while j != -1:
6              occ.append((i, j))
7              j = s.find(pattern, j+1)
8      return occ
9
10 myStringList = ['the rain in spain', 'ain\'t no sunshine',
11     'she was greeted with disdain']
12
13 occOfAin = findSubstringInStrings(myStringList, 'ain')
```

Quiz

Have you ever seen a function calling itself? Consider the following:

```
def fun(x):
    if len(x) > 1:
        return fun(x[1:])
    return x
```

What does the function call `fun([1,2,3,4])` return?

Quiz

Have you ever seen a function calling itself? Consider the following:

```
def fun(x):
    if len(x) > 1:
        return fun(x[1:])
    return x
```

What does the function call `fun([1,2,3,4])` return? [4]

Loops

Functions

Classes, Modules & Packages

Programming Errors & Debugging

Creating new types

- A `class` defines a new type
- It can provide
  - class variables & functions
  - instance variables & functions

Classes—example of code reuse

```
1  class Library:
2      description = 'This is a Library'
3  
4      def __init__(self, name):
5          # name the library
6          self.name = name
7          # create empty book storage on initialization
8          self.storage = list()
9  
10     def addBook(self, book):
11         self.storage.append(book)
12 
13     def getAllBooks(self):
14         return tuple(self.storage)
15 
16 myLib = Library('Bodleian Library')
17 myLib.addBook('The Art of Computer Programming (D. Knuth)')
```

Modules

- Every `.py` file is a module
- Modules can host functions, variables, and classes
- Imported modules with `import` statement
- Should not have blocks of code that are immediately executed
- Explicit reference to module scope: `global`
- Name of module available as global variable `__name__`

Modules—example of code reuse

mystringutils.py

```
1  #
2  # A module for all kinds of string utils
3  #
4
5  def findSubstringInStrings(stringCollection,
       pattern):
6      occ = list()
7      for i, s in enumerate(stringCollection):
8          j = s.find(pattern)
9          while j != -1:
10             occ.append((i, j))
11             j = s.find(pattern, j+1)
12     return occ
```

myscript.py

```
1  #!/usr/bin/env python3
2
3  import mystringutils
4
5  if __name__ == '__main__':
6      myStringList = ['the rain in spain',
7          'ain\'t no sunshine',
8          'she was greeted with disdain']
9
10     occOfAin = mystringutils.
           findSubstringInStrings(myStringList,
            'ain')
11     print(occOfAin)
```

Modules—example of code reuse

mystringutils.py

```
1   #
2   # A module for all kinds of string utils
3   #
4
5   def findSubstringInStrings(stringCollection,
        pattern):
6       occ = list()
7       for i, s in enumerate(stringCollection):
8           j = s.find(pattern)
9           while j != -1:
10              occ.append((i, j))
11              j = s.find(pattern, j+1)
12      return occ
```

myscript.py

```
1   #!/usr/bin/env python3
2
3   import mystringutils as su
4
5   if __name__ == '__main__':
6       myStringList = ['the rain in spain',
7           'ain\'t no sunshine',
8           'she was greeted with disdain']
9
10      occOfAin = su.findSubstringInStrings(
            myStringList, 'ain')
11      print(occOfAin)
```

Modules—example of code reuse

mystringutils.py

```
1   #
2   # A module for all kinds of string utils
3   #
4
5   def findSubstringInStrings(stringCollection,
        pattern):
6       occ = list()
7       for i, s in enumerate(stringCollection):
8           j = s.find(pattern)
9           while j != -1:
10              occ.append((i, j))
11              j = s.find(pattern, j+1)
12      return occ
```

myscript.py

```
1   #!/usr/bin/env python3
2
3   from mystringutils import
        findSubstringInStrings
4
5   if __name__ == '__main__':
6       myStringList = ['the rain in spain',
7           'ain\'t no sunshine',
8           'she was greeted with disdain']
9
10      occOfAin = findSubstringInStrings(
            myStringList, 'ain')
11      print(occOfAin)
```

Packages

- Way of structuring multiple modules into a directory hierarchy
- Package directories must contain a `__init__.py` file
- Can be imported the same way as modules
- Python itself offers many packages, and even more third-party packages are available through *package managers* such as `conda`

Quiz

- In Python, a class is ____________ for an object.

  a nuisance   an instance   a blueprint   a distraction

- Consider the following class:

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

What is the correct statement to instantiate a `Dog` object?

- `Dog('Rufus', 3)`
- `Dog(self, 'Rufus', 3)`
- `Dog.__init__('Rufus', 3)`

source (in part): https://realpython.com/quizzes

Quiz

- In Python, a class is ____________ for an object.

  a nuisance　　an instance　　a blueprint✔　　a distraction

- Consider the following class:

```
1 class Dog:
2     def __init__(self, name, age):
3         self.name = name
4         self.age = age
```

What is the correct statement to instantiate a `Dog` object?

  - `Dog('Rufus', 3)` ✔
  - `Dog(self, 'Rufus', 3)`
  - `Dog.__init__('Rufus', 3)`

source (in part): https://realpython.com/quizzes

Loops

Functions

Classes, Modules & Packages

Programming Errors & Debugging

Programming errors

Recognizing different types of errors:

- Syntactic: spelling & grammar mistakes
  - e.g. $avg = (x\,y)/2$
- Semantic: mistakes in meaning, context, or program flow
  - e.g. $avg = x + y/2$ or $avg = (x + z)/0$

Distinction between

- Compile-time errors (syntactic, semantic)
- Runtime errors (semantic)

RuntimeError

Changing the size of `my_dict` in loop

```
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     my_dict[(key, 1, 2, 3)] = 'new element'
```

Catching exceptions

Controlled treatment of anticipated exceptions:

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops!  That was no valid number.  Try again...")
```

Raising exceptions

Use `raise` keyword to throw exceptions:

```
def myFunction(collection):

    if len(collection) == 0:
        raise RuntimeError("Invalid input: empty collection")
    # do something ..
    return

myFunction(list())
```

Raising exceptions

Check properties of input parameters using the `assert` statement:

```python
def myFunction(collection):

    assert len(collection) > 0, "Invalid input: empty collection"

    # do something ..
    return

myFunction(list())
```

Failed assertions result in an `AssertionError`

Debugging

PDB—the Python debugger

- Enables step-by-step proceeding of statements in Python programs
- Interaction with Python program at runtime
- Debugger is invoked by *breakpoints*
- Set breakpoint in arbitrary location of your code by
  - calling builtin "`breakpoint()`" function (Python version $\geq$ 3.7)
  - statement "`import pdb; pdb.set_trace()`"

Python debugger—example

```
1  # dictionary filled with arbitrary elements
2  my_dict = {'key': 'value', 1: 'text', (1, 2)
      : 'text'}
3
4  # invoke Python debugger
5  breakpoint()
6
7  # for-loop over keys of my_dict with control
       variable 'key'
8  for key in my_dict:
9      my_dict[(key, 1, 2, 3)] = 'new element'
```

Quiz

- Is improper indentation a syntactic or semantic error?
- Consider the following code:

```
1 def str2int(x):
2     try:
3         return int(x)
4     __________ ValueError:
5         return -1
```

What keyword should be used here?

`except` `raise` `else` `Exception`

source: https://quizizz.com/

Quiz

- Is improper indentation a syntactic or semantic error? syntactic
- Consider the following code:

```
1 def str2int(x):
2     try:
3         return int(x)
4     __________ ValueError:
5         return -1
```

What keyword should be used here?

except✔   raise   else   Exception

source: https://quizizz.com/

Recap

Summary

- `for` and `while`
- Code reuse through
  - Functions
  - Classes
  - Modules & Packages
- Compile-time and runtime errors
- Python debugger, a tool for hunting runtime errors (bugs)