Deep Learning

Übung WS 23/24

Judith Nester (nester@uni-koeln.de)

02-11-2023

Recap

- Python
  - Running Python scripts
  - Python syntax
  - Data types
  - Standard library
- Exercise 2

Recap

» Are there any questions about the reference solution?

Today

List Comprehension

Functions

Input/Output

Exercise

⚠ Slides will only scratch the surface, reading documentation is necessary!

Section 1

List Comprehension

List Comprehension

- Python-specific!
- Special syntax for specifying lists in just one line of code
  - KISS principle
- Similar to set-builder notation
- Why is list comprehension important for Deep Learning?
  - Preparation and shaping of data

List Comprehension

- Python-specific!
- Special syntax for specifying lists in just one line of code
  - KISS principle
- Similar to set-builder notation
- Why is list comprehension important for Deep Learning?
  - Preparation and shaping of data

Lists in Python

```
1 # an empty list
2 x = []
3 # a list with two elements
4 x = [1, 2]
5 # a list with three elements of different types
6 x = [1, True, "Hello"]
```

List Comprehension

- Python-specific!
- Special syntax for specifying lists in just one line of code
  - KISS principle
- Similar to set-builder notation
- Why is list comprehension important for Deep Learning?
  - Preparation and shaping of data

Lists in Python

```
1 # an empty list
2 x = []
3 # a list with two elements
4 x = [1, 2]
5 # a list with three elements of different types
6 x = [1, True, "Hello"]
```

Enumeration of list elements

List Comprehension

Examples

```
1 l1 = [1,2,3]
2 l2 = [x*2 for x in l1] # [2,4,6]
```

List Comprehension

Examples

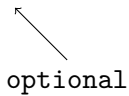
```
l1 = [1,2,3]
l2 = [x*2 for x in l1] # [2,4,6]

l1 = ["the", "dog", "barks"]
l2 = [x for x in l1 if "e" in x] # ["the"]
```

List Comprehension

Formal

```
[ expr for x in list for/if ... ]
```



- Expression `expr` is evaluated in context of `for` loop
  - Expression can be full-fledged python expression
    e.g., another list comprehension!
- One `for` loop is required, more are optional
- `expr` often uses `x`, but doesn't have too
- `list` needs to be a list (or coercible into one)
- Evaluates to a list (and the whole thing is an expression)

List Comprehension

Demo

Section 2

Functions

Functions

- Sub-programs in your program
- Built-in functions: `print(...)`, `input(...)`, `str(...)`, ...
- Custom functions: Defined by yourself

Functions

- Sub-programs in your program
- Built-in functions: `print(...)`, `input(...)`, `str(...)`, ...
- Custom functions: Defined by yourself

```
1 def myFunction(arguments):
2     # execute code
3     return
```

Function Arguments

- Arguments are (like everything else) dynamically typed
- Multiple arguments are listed with a comma

Function Arguments

- Arguments are (like everything else) dynamically typed
- Multiple arguments are listed with a comma

```
1 def myFunction(arg1, arg2, arg3):
2     # do something
3     return
```

Function Arguments

Type checking and default values

Listing: Type Checking

```
def myFunction(arg1, arg2):
    if not isinstance(arg1, str):
        # throw exception
        raise TypeError
    # do something
    return
```

Function Arguments

Type checking and default values

Listing: Type Checking

```
1 def myFunction(arg1, arg2):
2     if not isinstance(arg1, str):
3         # throw exception
4         raise TypeError
5     # do something
6     return
```

```
1 def myFunction(arg1, arg2="bla"):
2     print(arg2)
3     return
```

Function Arguments

Named arguments

Argument names can be used when calling a function

```
1 def myFunc(a1, a2):
2     # do something
3     return
4
5 myFunc(1,2) # a1 = 1, a2 = 2
6 myFunc(a2=1, a1=2) # a2 = 1, a1 = 2
```

Variable Arguments

Argument values as list

We can define functions without a fixed number of arguments:

```
1 def myFunc(*args):
2     # Inside the function, args is a list
3     numArgs = len(args)
4     print(str(numArgs) + " arguments were given.")
5     print("the first argument is " + str(args[0]))
6     return
7
8 myFunc(1,2,3)
9 myFunc(1,2)
```

Variable Arguments

Named arguments as dictionary

```
1 def func(**kwargs):
2     # inside the function, kwargs is a dictionary
3     for key in kwargs:
4         print(key + ": " + str(kwargs[key]))
5     return
6
7 func(a=1, b="Hallo", c=7)
```

Terminating a Function

» We can terminate the function with the `return` statement

Terminating a Function

» We can terminate the function with the `return` statement

```
1 # return value is 5
2 def f1():
3     return 5
```

Terminating a Function

» We can terminate the function with the `return` statement

```
1 # return value is 5
2 def f1():
3     return 5
4
5 # return value is None
6 def f2():
7     return
```

Terminating a Function

» We can terminate the function with the `return` statement

```
1 # return value is 5
2 def f1():
3     return 5
4
5 # return value is None
6 def f2():
7     return
8
9 # return value is None
10 def f3():
11     return None
```

Terminating a Function

» We can terminate the function with the `return` statement

```
1  # return value is 5
2  def f1():
3      return 5
4
5  # return value is None
6  def f2():
7      return
8
9  # return value is None
10 def f3():
11     return None
12
13 # return value is None
14 def f4():
15     x = 3
```

Terminating a Function

» We can terminate the function with the `return` statement

```
# return value is 5
def f1():
    return 5

# return value is None
def f2():
    return

# return value is None
def f3():
    return None

# return value is None
def f4():
    x = 3
```

```
# return value is a list
def f5():
return [5]
```

Terminating a Function

» We can terminate the function with the `return` statement

```python
# return value is 5
def f1():
    return 5

# return value is None
def f2():
    return

# return value is None
def f3():
    return None

# return value is None
def f4():
    x = 3
```

```python
# return value is a list
def f5():
return [5]

# return value is a dictionary
def f6():
return {"value": 5}
```

Terminating a Function

» We can terminate the function with the `return` statement

```python
# return value is 5
def f1():
    return 5

# return value is None
def f2():
    return

# return value is None
def f3():
    return None

# return value is None
def f4():
    x = 3
```

```python
# return value is a list
def f5():
return [5]

# return value is a dictionary
def f6():
return {"value": 5}

# return value is a list
def f7():
return [None]
```

Terminating a Function

» We can terminate the function with the `return` statement

```python
# return value is 5
def f1():
    return 5

# return value is None
def f2():
    return

# return value is None
def f3():
    return None

# return value is None
def f4():
    x = 3
```

```python
# return value is a list
def f5():
return [5]

# return value is a dictionary
def f6():
return {"value": 5}

# return value is a list
def f7():
return [None]

# an empty function body also returns
# None
def f8():
pass
```

None

- `None` is the null-value in Python
- The type of `None` is `NoneType`
- `NoneType` only has a single possible value: `None`

None

- `None` is the null-value in Python
- The type of `None` is `NoneType`
- `NoneType` only has a single possible value: `None`
- If evaluated, `None` is considered to be false

```
if None:
    # Never executed
    print("Nothing is false")
```

Section 3

Input/Output

Introduction

- IO: Input/Output
- Standard model: Streams of bytes/characters: Most programming languages

Introduction

- IO: Input/Output
- Standard model: Streams of bytes/characters: Most programming languages
- Regular workflow
  1. Open a file
     - Configure: Mode, encoding, file type, ...
  2. Read from file or write to file
  3. Close file

IO in Python

Listing: File Reading

```
# open file and create a file
# object
fh = open(FILENAME)

# call a method on file object
for line in fh.readlines():
    # process each line

# close object
# (important, because open files
# consume resources)
fh.close()
```

Listing: File Writing

```
# open file and create a file
# object
fh = open(FILENAME, mode="w")

# write list content into file
for line in listOfResults:
    fh.write(line)

# close object
# (important, because open files
# consume resources)
fh.close()
```

IO in Python

open()

```
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)
```

- https://docs.python.org/3/library/functions.html#open
- mode: Reading or writing
- buffering: Controls when and in which chunks data is actually written to the disk
- encoding: Specify encoding (hint: Make sure that UTF-8 is used)
- errors: How to handle encoding errors
- newline: Which characters are used to indicate a new line?
- ...

IO in Python

Methods of the File Object

- `https://docs.python.org/3/library/io.html#module-io`
- `close()`: Flush and close the stream
  - In Java, `close()` only closes without flushing
- `flush()`: Flush the stream
- `readline(size=-1)`: Read and return one line from the stream
- `readlines(hint=-1)`: Read and return a list of lines from the stream.
- `write(s)`: Write the string s to the stream and return the number of characters written.
- ...

Section 4

Exercise

Exercise 03

https://github.com/IDH-Cologne-Deep-Learning-Uebung/exercise-03