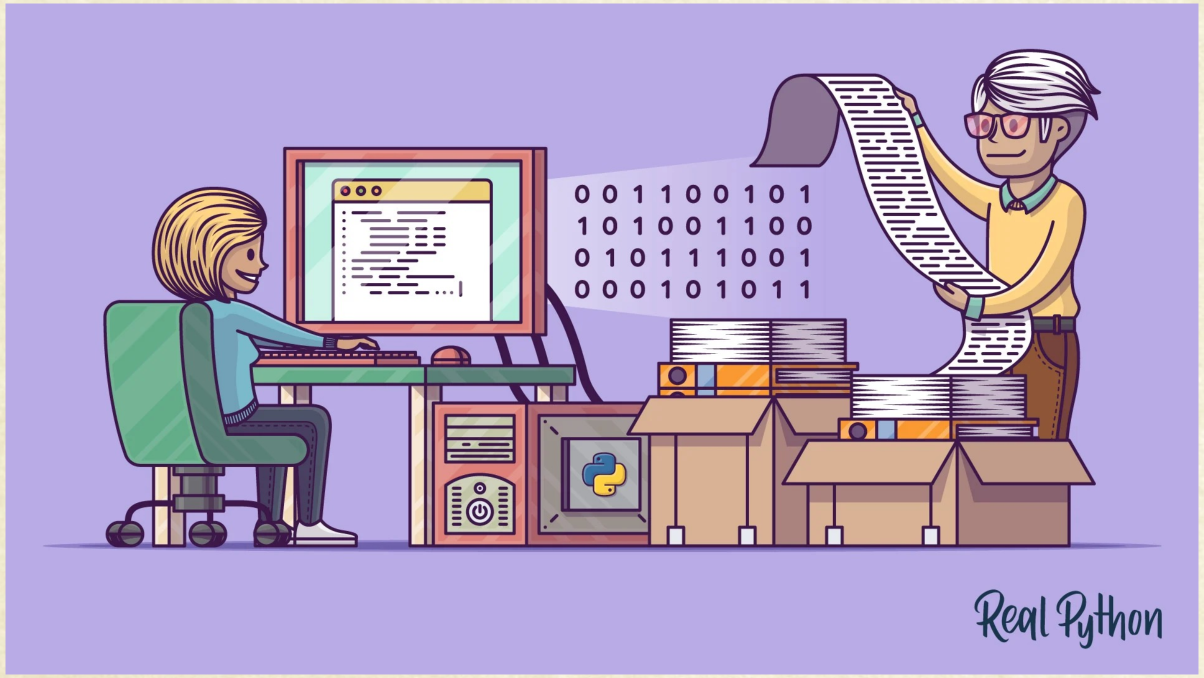
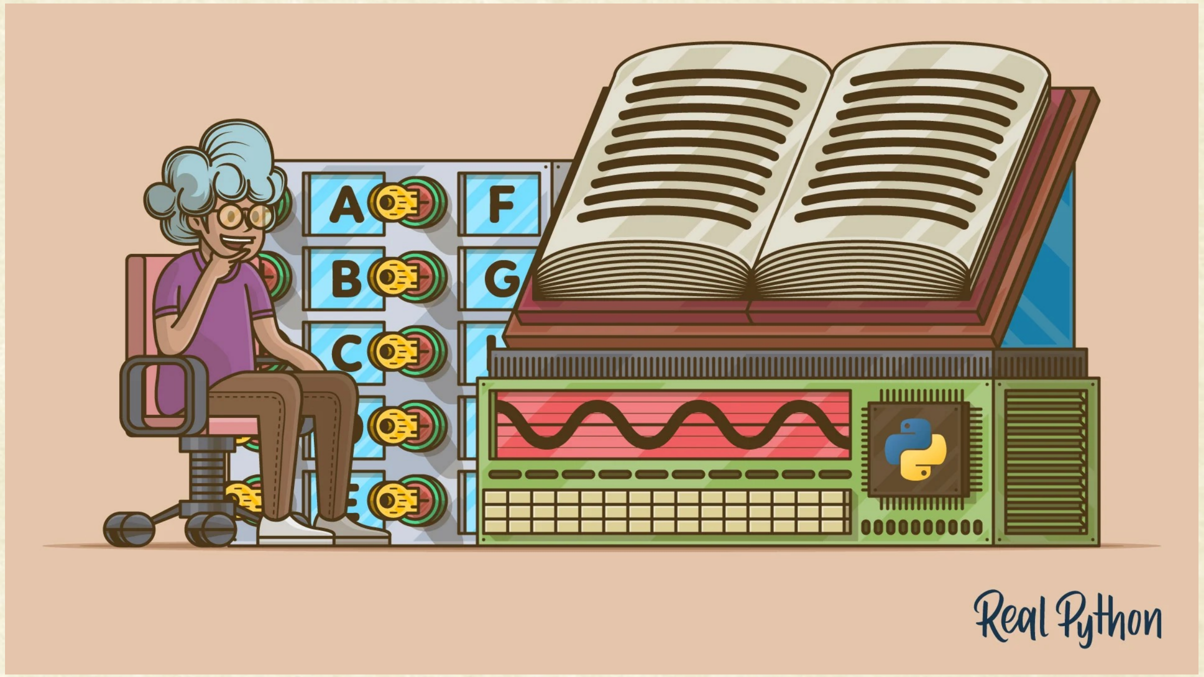
Lecture 6: Files, Tuples, Dictionaries





Anoop M. Namboodiri

IIIT Hyderabad

Python Files

Reading / Writing files

File Object

- f = open('filename.txt', ‘rw’)
  - Creates a file object, f.
  - Modes: read, write, append.
- f.close()
  - Should close after the need
  - Can run out of file descry
- f.seek(0)
  - Go to beginning, end, pos.
- f.write(str)
  - Write str to current pos
- data=f.read(numChars)
  - Read numChars/whole file
  - pos changes to numChars
- f.write(str)
  - Write str to current pos
- File object variables:
  - print(f.name)
  - print(f.mode)

Looping through its contents

```
f = open('filename.txt', 'r')
for line in f:
    print(line)
f.close()
```

```
with open('filename.txt', 'r') as f:
    lines = f.readlines()
    for line in lines:
        print(line)
```

```
with open('filename.txt', 'r') as f:
    for line in f:
        print(line)
```

```
with open('filename.txt', 'r') as f:
    data = f.read()
    words = data.split()
    for word in words:
        print(word)
```

List Comprehensions

A Map-Filter Construct

List Comprehension

- Assume you want to perform an operation on each element of an existing list and create a ***newlist*** from the results.
- We want to create a list of fruits that contain the letter ‘a’

```
fruits = ["apple", "banana", "cherry", ”plum", "mango"]

newlist = []
for x in fruits:
  if ‘a’ in x:
    newlist.append(x)

print(newlist)
```

List Comprehension

- List Comprehension allows you to define a mapping that is applied to every element in a list

```
fruits = ["apple", "banana", "cherry", ”plum", "mango"]
newlist = [x.capitalize() for x in fruits if ‘a’ in x]
print(newlist)
```

- Avoids the for-loop and hence is faster too.
- Think of it as a mapping-filter operation
- Also see the map() function.

Python Tuples

The immutable sequence

Tuples: The Immutable One

- Creating
  - tuple1 = (elt1,elt2, .., eltn) or
  - tuple1 = elt1, elt2, .., eltn
- Element access using tuple1[ ], just as in lists,
- Tuples once created, cannot be changed
  - tuple1[1] = eltm will throw an error
- However, mutable elements could be modified
  - If tuple1[2] is a list, tuple1[2].append(eltm) is fine
- Elements need not be of the same type.
- Like strings: Indexing, Slicing, Concatenation
- zip() to create tuples from two lists

Why Tuple over List

- Conceptually or by convention
  - Lists is a homogenous collection, while tuple is a structured, heterogeneous, sequence of elements.
  - Positions have meaning in tuple; like (latitude, longitude)
  - Use tuple to indicate constant sets (days of a week)
- Tuples with immutable elements can be used as keys of a dictionary
- Tuples take slightly lesser space compared to lists
  - Important only for large collections of data
- Think of tuples like struct in C.

Let us Code: Tuple

- Create two tuples, dayname, and dom, to hold the names of days and number of days in month.
- The program should print the day, given a date
- Assume Year = 2024; Jan 1 was a Monday

```
dayname = ('Sunday','Monday','Tuesday',\
'Wednesday','Thursday','Friday','Saturday')

dom = (31,29,31,30,31,30,31,31,30,31,30,31)

DD,MM = input('Enter DD,MM : ').split(',')

days = sum(dom[:int(MM)-1]) + int(DD)

print(DD,MM," is a ",dayname[days%7])
```

Python Dictionaries

The Mapping

Dictionaries: The Mapping

- Creating from key:value pairs or from tuple list:
  - dict1 = {key1:val1, key2:val2, .., keyn:valn}
  - dict1 = dict((key1,val1),(key2,val2), .., (keyn,valn))
- Element access using dict1[keyx], just as in lists
- New key:val pairs may be inserted or existing values may be modified after creation; both by:
  - dict1[keyx] = valx
- Elements or keys need not be of the same type.

Let us Code: Dictionary

- Write a program to compute the frequency of occurrence of each character in a text file (unigram)

```
unigram = {}
f = open(”inp1.txt", 'r')
for line in f:
     for char in list(line):
          if char in unigram:
               unigram[char] += 1
          else:
               unigram[char] = 1

for key in sorted(unigram):
     print(key,":", unigram[key])
```

Let us Code: collections.defaultdict

- Several containers in collections
- defaultdict allows us to access/use a field without initializing it

```
from collections import defaultdict
unigram = defaultdict(int)
f = open(”inp1.txt", 'r')
for line in f:
     for char in list(line):
          unigram[char] +=  1

print(sorted(unigram.items(), key=lambda item: item[1]))
```