COMPUTER PROGRAMMING RECAP

Master in Environmental Management of Mountains Areas

ADVANCED GEOMATICS

Andrea Antonello - Free University of Bolzano

March - June 2024

LISTS

OPERATIONS ON LISTS

- create a list of strings
- elements of a list can be accessed through the array notation using a 0 based index
- add an element to the end of the list
- remove an element by object
- remove an element by index

```
mylist = ["Merano", "Bolzano", "Trento"]
print(mylist)
print("The elements start at position 0: " + mylist[0])

mylist.append("Potsdam")
print(mylist)
mylist.remove("Potsdam")
print(mylist)
mylist.pop(0)
print(mylist)
```

CHECK IF AN ELEMENT IS IN A LIST

To check if an element is in a list, the **'in'** operator is used:

```python
mylist = ["Merano", "Bolzano", "Trento"]

doIHaveBolzano = "Bolzano" in mylist
print(doIHaveBolzano)

doIHavePotsdam = "Potsdam" in mylist
print(doIHavePotsdam)
```

LOOPING OVER LISTS

To loop over lists the **'for'** construct is used:

```python
for item in iterable:
    print(item)
```

if an index is necessary, we can use a **'range'** and the **'len'** function:

```python
colors = ["red", "green", "blue", "purple"]
ratios = [0.2, 0.3, 0.1, 0.4]
for index in range(len(colors)):
    ratio = ratios[index]
    color = colors[index]
    print(f"{color} -> {ratio}")
```

BREAK AND CONTINUE

The **break** statement is used to exit a loop early,

```
for i in range(10):
    if i == 5:
        break
    print(f"A) {i}")
```

while the **continue** statement is used to skip the rest of the code inside the loop for the current iteration.

```
for i in range(10):
    if i == 5:
        continue
    print(f"B) {i}")
```

A WORD ABOUT RANGES

Ranges produce a sequence of consecutive integers.

Loop over a range from 0-10. Mind that the last number is not included.

```
for i in range(0,10):
    print(f"A) {i}")
```

The initial value is optional and defaults to 0.

```
for i in range(10):
    print(f"B) {i}")
```

A step can be set, here we print every second number:

```
for i in range(0,10,2):
    print(f"C) {i}")
```

And we can use a negative step to do the same descending:

```
for i in range(10,0,-2):
    print(f"D) {i}")
```

SORTING LISTS

```python
mylist = ["Merano", "Bolzano", "Trento"]
print( f"This is the original mylist: {mylist}" )

mylist.sort()
print( f"This is the sorted mylist: {mylist}" )

mylist.sort(reverse = True)
print( f"This is the reverse mylist: {mylist}" )
```

The **sort** method has a **key** parameter used to specify a function to be called on each list element prior to making comparisons.

```python
mylist = ["banana", "Orange", "Kiwi", "cherry"]
mylist.sort()
print(f"A mixed case mylist, sorted: {mylist}")

mylist.sort(key = str.lower)
print(f"A mixed case mylist, properly sorted: {mylist}")

numlist = ["002", "01", "3", "004"]
def toInt(string):
    return int(string)

numlist.sort(key = toInt)
print(f"A formatted list of nums, properly sorted: {mylist}")
```

LAST ABOUT LISTS

- lists can be merged with the plus operator
- lists can be concatenated to a string using **separator.join(list)**
- in case of numbers some functions apply

```
abc = ["a", "b", "c"]
cde = ["c", "d", "e"]
newabcde = abc + cde
print( newabcde )

print( ";".join(newabcde) )
print( " | ".join(newabcde) )

nums = [1.0, 2, 3.5]
print( max(nums) )
print( min(nums) )
print( sum(nums) )
```

DICTIONARIES

WHAT ARE THEY?

A Hashmap or Dictionary is a container of key and value pairs.

Think of it as an actual dictionary, where you have **definitions (the value**) stored under certain **names (the key**).

So you can ask the dictionary for the definition of using the name. Mind that names/keys are case sensitive.

Also keys are unique, so you can’t have two definitions for the same name. **If you insert a new value for an existing key, the old value is overwritten.**

CREATE, GET, ADD, REMOVE

- get a value from the dictionary through its key
- add a new key/value pair to an existing dictionary
- remove a key/value pair

```
townsProvinceMap = {
    "merano":"BZ", "bolzano":"BZ", "trento":"TN"
}

print(townsProvinceMap["merano"])
townsProvinceMap["Potsdam"] = "BR"
print(townsProvinceMap)
townsProvinceMap.pop("Potsdam")
print(townsProvinceMap)
```

WHAT IF AN ITEM DOESN’T EXIST?

If you try to access a key that doesn’t exist, you will get a KeyError. To avoid this, you can use the get method, which will return None if the key doesn’t exist.

```
if townsProvinceMap.get("Merano") is None:
    print("The key doesn't exist")
else:
    print("The key exists")
```

It is also possible to provide a default value to the get method, which will be returned if the key doesn’t exist.

```
print( townsProvinceMap.get("merano", "unknown") )
```

LOOPING DICTIONARIES

Remember that a dictionary item is a key/value pair, so we need 2 variables, but apart of that, looping is the same as for lists.

```
for key, value in townsProvinceMap.items():
    print( key + " is in province of " + value )
```

KEYS AND VALUES

Dictionaries have methods to get the keys and values.

```
print( townsProvinceMap.keys() )
print( townsProvinceMap.values() )
```

In python, dictionaries are ordered following the insertion order. If sorting by key is needed, the best way to do so is to sort the keys and loop over them.

Since the **keys()** method returns an iterable, we can’t directly sort it (even if we can loop over it). We need to convert it to list first.

```
towns = list(townsProvinceMap.keys())
towns.sort()
for town in towns:
    print( town + " is in province of " + townsProvinceMap[town] )
```

A PATTERN YOU NEED TO LEARN

Due to the uniqueness of their keys, dictionaries are often used to count objects in datasets or aggregate them.

```python
myText = """
    We would like to know how many times
    every character appears in this text.
"""

charDictionary = {}
for character in myText.strip():
    count = charDictionary.get(character, 0)
    count += 1
    charDictionary[character] = count

for key, value in charDictionary.items():
    if key == " ":
        key = "The space"
    elif key == "\n":
        key = "The newline"
    print(key, "appears", value, "times.")
```

HANDLING TEXT FILES

WRITING A TEXT FILE

Let’s write some data into a text file

```python
filepath = "..../data.txt"
data = """# station id, datetime, temperature
1, 2023-01-01 00:00, 12.3

2, 2023-01-01 00:00, 11.3
3, 2023-01-01 00:00, 10.3"""


with open(filepath, "w") as file:
    file.write(data)
```

If you want to append to an existing file, you can use the "a" mode instead of "w".

```python
with open(filepath, "a") as file:
    file.write("\n1, 2023-01-02 00:00, 9.3")
    file.write("\n2, 2023-01-02 00:00, 8.3")
```

READING A TEXT FILE

Let’s read the file, parse it and count the occurrences of each station id.

```python
with open(filepath, "r") as file:
    lines = file.readlines()

stationCount = {}
for line in lines:
    line = line.strip()
    if line.startswith("#") or len(line) == 0:
        continue
    stationId = line.split(",")[0]
    count = stationCount.get(stationId, 0)
    count += 1
    stationCount[stationId] = count

for key, value in stationCount.items():
    print(f"Station {key} appears {value} times.")
```

```
<license>
  This work is released under Creative Commons Attribution
  Share Alike (CC-BY-SA).
</license>
<sources>
  Much of the knowledge needed to create this training material has
  been produced by the sparkling knights of the
  <a href="http://www.osgeo.org">OSGEO</a> and
  <a href="http://qgis.org">QGIS</a>,
  communities.
  Their websites are filled up with learning material that can be used
  to grow knowledge beyond the boundaries of this lessons
  Another essential source has been the Wikipedia project.
</sources>
<important>
  This work is part of the Advanced Geomatics Course given in 2024 at the
  EMMA Master of the Free University of Bolzano.
</important>
```