Useful python commands for introduction to programming and dataprocessing

Christian E.L. Madsen, s140279

2020/12/29

Table of Contents

Conversions

string → list	list(string)
list → string	' '.join(list)
list → array	np.array(list)
array → list	list(array)
float → int	int(float)
int → float	float(int)
char → int	ord(char)
int → char	chr(int) height

The generic command for list → string is 'X'.join(list) where X can be anything and is put between each element in the list.
An example could be: 'X'.join([a,b,c]) → aXbXc

To split a string around certain elements, you can use string.split('X') where X can be anything and is not included in the result, for instance:
"Text with words split by space".split(' ') → ["Text","with","words","split","by","space"]

Commands for Lists

myList = list() z = myList.pop() cell7	Creates an empty list with name myList Saves the last value from myList in z and deletes it from myList. cell8

Commands for Arrays and matrixes

myArr = someArr==Value
Returns an array of True and False depending on if the individual elements in someArr are equal to value. Also useable with other equalities like $<=, <, != $ etc.

myArr = myArr[myArr<Value]
Returns an array that only contains the elements in myArr that are less than Value.

z = np.delete(x,index,axis)
Saves a copy of x in z with the row/column removed at the specificied index. Axis 0 is rows/horizontal, 1 is columns/vertical.

Dictionaries, sets and tuples

Creating a tuple: myTuple = ((A,B,...,N),(A1,B1,...,N1),(A2,B2,...N2))

Elements in tuples don't need same length. Useful for grouping values tied to the same element together.
To access value A1, use myTuple[1][0], because it is element 1's value 0

```
def computeAgeGroup(age):
    ageGroups = ((0,"Infant",1),(1,"Toddler",3),(3,"Child",13),
                 (13,"Teenager",20),(20,"Adult",float("inf")))
    for i in range(len(ageGroups)):
        if age >= ageGroups[i][0] and age < ageGroups[i][2]:
            ageGroup = ageGroups[i][1]
    return ageGroup
```

Figure 1: Code example using a tuple to tie together lower and upper bounds on age with a string describing the ageGroup

Creating a dictionary: DICT = {From:To,From1:To1,From1:To1,From2:To2}

DICT[z] and DICT.get(z) used to look for z in the dictionary and translate and return it. Returns an error if nothing is found.
A more versatile command is DICT.get(z,x). This tries to translate z, if z isn't found in the dictionary, x is returned instead.

```
####
RELATIONS = {'A':2,'B':2,'C':2,'D':3,'E':3,'F':3,
             'G':4,'H':4,'I':4,'J':5,'K':5,'L':5,
             'M':6,'N':6,'O':6,'P':7,'Q':7,'R':7,
             'S':7,'T':8,'U':8,'V':8,'W':9,'X':9,'Y':9,'Z':9}
def alphaToPhone(alpha):
    return ''.join([str(RELATIONS.get(elements,elements)) for elements in list(alpha)])
```

Figure 2: Code example using a dictionary to convert a partially converted string of letters and numbers. Since some numbers were already converted, we used .get(a,b) to avoid errors.

Creating a set: mySet = {'val1','val2','val3','val4'}

Sometimes useful in if statements where we need to act on circumstances not easy to explain to a computer or would requite a lot of "or".

```
vowelSet = {'a','e','i','o','u','y'}
def syllables(word):
    charList = list(word)
    vowelFound = False
    syl = 0
    for char in charList:
        if char in vowelSet:
            vowelFound = True
        elif vowelFound:
            if char not in vowelSet:
                syl += 1
                vowelFound = False
        else:
            syl +=0
    return syl
```

Figure 3: Code example using a set of vowels to branch the if statements