COMP 204

Operations on containers: enumerate, zip, comprehension

Mathieu Blanchette
based on material from Yue Li, Carlos Oliver Gonzalez and Christopher Cameron
There exist many ways to format strings for printing (Section 7.1). **Formatted String Literals** are very useful:

```python
pi = 3.1415927

# standard printing
print('pi is', pi)

# printing using formatted strings
print(f'pi is {pi}')
print(f'pi is approx. {pi:.3f}') # to round to 3 decimals

grades = {'Sjoerd': 8, 'Jack': 74, 'Annie': 100}
for name, grade in grades.items():
    # prints name over 10 characters, and grade over 5
    print(f'{name:10} ===> {grade:5d}')
```

#output:
# pi is 3.1415927
# pi is 3.1415927
# pi is approx. 3.142
# Sjoerd ===> 8
# Jack ===> 74
# Annie ===> 100
Today, we introduce convenient Python techniques that simplify our code and (sometimes) make it more efficient.

- **enumerate**: Loop through lists keeping track of index of items
- **zip**: Loop through multiple lists in parallel
- **Comprehension**: Construct new lists, sets, or dictionaries from existing ones.

**Important**: What we can do with enumerate, zip, and comprehensions can always be done with standard for loops. These techniques just make it easier.
A very common thing when dealing with lists is to iterate over each index and doing some computation with each element.

```python
L = some_list
for index in range(len(L)):
    item = L[index]
    # do something with item and index
```

The `enumerate` function allows to do this more simply:

```python
L = some_list
for index, item in enumerate(L):
    # do something with item and index
```

Note: You can always use a loop over indices (as above) instead of a loop with `enumerate` (as below). The second is just simpler and more efficient.
Enumerate - examples

Goal: Iterate through a list of names and print each name and the index at which it is located.

```python
names = ["Hillary","Yang","Bernard","Drina"]

# Goal: Print each name and its index in the list

# using for loop over indices
for index in range(len(names)):
    name = names[index]
    print(name,"is at index",index)

# using enumerate
for index, name in enumerate(names):
    print(name,"is at index",index)
```
Goal: Iterate through a list of names and print those whose age is below 18.

```python
names = ['Hillary', 'Yang', 'Bernard', 'Drina']
ages = [42, 15, 23, 17]  # the age of each person

# Goal: Print the name of all people below 18 years old

# using for loop over indices
for index in range(len(names)):
    name = names[index]
    if ages[index] < 18:
        print(name, 'is a minor')

# using enumerate
for index, name in enumerate(names):
    if ages[index] < 18:
        print(name, 'is a minor')
```
**Zip**

Often, we need to iterate over the elements of two lists in parallel (as in our previous example).

```python
A = some_list
B = some_other_list
for index in range(len(A)):
    item_A = A[index]
    item_B = B[index]
    # do something with item_A and item_B
```

The **zip** function allows to do this more simply:

```python
A = some_list
B = some_other_list
for item_A, item_B in zip(A, B):
    # do something with item_A and item_B
```

**Notes:**

- If list B is shorter than list A, we get an error.
- Zip also works with more than two lists.
Example: Assemble list of full names from list of names and list of surnames

```python
names = ['John', 'Daenery', 'Jamie', 'Tyrion', 'Robert']
surnames = ['Snow', 'Targaryen', 'Lannister', 'Lannister', 'Baratheon']

# without the zip function, assembling full names
# is a bit complicated
full_names = []
for index in range(0, len(names)):
    full_names.append(names[index] + '' + surnames[index])
print(full_names)

# or
full_names = []
for index, first in enumerate(names):
    full_names.append(first + '' + surnames[index])
print(full_names)

# This is easier to do with the zip function
full_names = []
for first, last in zip(names, surnames):
    full_names.append(first + '' + last)
print(full_names)
```
Zip - example

Zip can operate on more than two lists.

Example: Print the season where each character dies

```python
names = ['John', 'Daenery', 'Jamie', 'Tyrion', 'Robert']
surnames = ['Snow', 'Targaryen', 'Lannister', 'Lannister', '
            Baratheon']
deads = [5, 8, 8, None, 1]

for first, last, death in zip(names, surnames, deaths):
    print(first + " " + last + " dies in season " + str(death))
```
List comprehension

Very often, we need to assemble a list of objects based on iterating through and processing another list of objects.

```
L = some_list
result = []

for item in L:
    new_object = some_expr(item)
    result.append(new_object)
```

List comprehension allows doing this in a simple and efficient manner.

```
L = some_list
result = [ some_expr(item) for item in L ]
```
List comprehension - example 1

# Given a list of length of genes (nucleotides),
# Produce list of length of proteins (amino acids)
length_of_gen = [160, 393, 3012, 192, 27]

# with standard for loop
length_of_proteins = []
for n in length_of_gen:
    length_of_proteins.append(n/3)

# using list comprehension
length_of_proteins = [n/3 for n in length_of_gen]
# Produce the list of the squares of integers from 0 to 100

# with a standard for loop
squares = []
for n in range(101):
    squares.append(n*n)

# with list comprehension
squares = [ n*n for n in range(101) ]
# Given a gene sequence (starting with a start codon),
# Produce the list of amino acids it corresponds to
# Assume that you have a function aminoacid() that returns
# the amino acids encoded by a certain codon
s="ATGCAGCATGAAGATGAA"

# with a for loop:
aa_list =[]
for i in range(0,len(s),3):
    aa_list.append( aminoacid(s[i:i+3]) )

# with list comprehension:
aa_list= [ aminoacid(s[i:i+3]) for i in range(0,len(s),3) ]

# Note: to join all the aa in aa_list into a single string:
aa_string = "".join( aa_list )
List comprehension with conditional

Often, we want to make the inclusion in the result list conditional on some property of the item.

```python
L = some_list
result = []

for item in L:
    if some_test(item):
        new_object = some_expr(item)
        result.append(new_object)
```

List comprehension allows doing this in a simple and efficient manner.

```python
L = some_list
result = [some_expr(item) for item in L if some_test(item)]
```
# Goal: Produce a list of the squares of all odd numbers between 0 and 100

with for loop

```python
squares_odd = []
for n in range(101):
    if n%2 == 1:
        squares_odd.append(n*n)
```

with list comprehension

```python
squares_of_odd = [i*i for i in range(101) if i%2==1]
```
# Goal: Produce a list of character names that contain the letter "N"

names = ['John', 'Daenerys', 'Jamie', 'Tyrion', 'Robert']

# with for loop
names_with_N = []
for name in names:
    if 'n' in name or 'N' in name:
        names_with_N.append(name)

# with list comprehension
names_with_N = [name for name in names
                if 'n' in name or 'N' in name]
# Goal: Produce a list of the full names of all members of the Lannister family

names = ['John', 'Daenery', 'Jamie', 'Tyrion', 'Robert']
surnames = ['Snow', 'Targaryen', 'Lannister', 'Lannister', 'Baratheon']

# with for loop
lannisters = []
for name, surname in zip(names, surnames):
    if surname == 'Lannister':
        lannisters.append(name)

# with list comprehension
lannisters = [name + ' ' + surname for name, surname in zip(names, surnames) if name == 'Lannister']
Set comprehension

We can use comprehension to build a set, in a manner similar to list comprehension, but using `{}` instead of `[]`.

```python
# Goal: Produce a Set of family surnames for which at least one family member is still alive at the end of season 7

names = ['John', 'Daenerys', 'Jamie', 'Tyrion', 'Robert']
surnames = ['Snow', 'Targaryen', 'Lannister', 'Lannister', 'Baratheon']
dead = [5, 8, 8, None, 1]

# with for loop
alive = set([]) # empty set
for surname, death in zip(surnames, dead):
    if death == None or death >= 8:
        alive.add(surname)

# with list comprehension
alive = {surname for surname, death in zip(surnames, dead) if death == None or death >= 8}
```
Dictionary comprehension

We can use comprehension to build dictionaries.
With a standard for loop:

```python
D = some_dictionary
result = []

for k, v in D.items():
    if some_test(k, v):
        new_key = some_key_expr(k, v)
        new_value = some_value_expr(k, v)
        result[new_key] = new_value
```

With dictionary comprehension:

```python
D = some_dictionary
result = {some_key_expr(k, v): some_value_expr(k, v) for k, v in D if some_test(k, v)}
```
Dictionary comprehension - Example 1

# Goal: Given a dict. of keys=names, values=(height, weight)
# Produce: a dict. of keys=names, values=BMI, which
# includes only patients whose BMI is above 30

def BMI(h,w):
    return w/(h*h)

patient_dict={"John":(1.6,70),"Daenerys":(1.5,55),"Jamie":(1.8,85),"Tyrion":(1.0,40),"Robert":(1.8,140)}

# with a for loop
high_BMI={}
for name,(h,w) in patient_dict.items():
    bmi=BMI(h,w)
    if bmi>30:
        high_BMI[name]=bmi
print(high_BMI)

# with a dictionary comprehension
high_BMI = {name:BMI(h,w) for name,(h,w) in patient_dict.items() if BMI(h,w)>30}
print(high_BMI)