**List methods**

Let's practice list methods!

Let's imagine a situation: you went to the market and filled your baskets (basket1 and basket2) with fruits. You wanted to have one of each kind but realized that some fruits were put in both baskets.

**Task 1**. Your first task is to remove everything from basket2 that is already present in basket1.

**Task 2**. After the removal it is reasonable to anticipate that one of the baskets might weigh more compared to the another (all fruit kinds weight the same). Therefore, the second task is to transfer some fruits from a heavier basket to the lighter one to get approximately the same weight/amount of fruits.

**Instructions 1/2**

**50 XP**

* [1](about:blank)
* [2](about:blank)
* Remove fruits from basket2 that are already present in basket1.

# Remove fruits from basket2 that are present in basket1

for item in basket1:

if item in basket2 :

basket2.remove(item)

print('Basket 1: ' + str(basket1))

print('Basket 2: ' + str(basket2))

<script.py> output:

Basket 1: ['banana', 'kiwifruits', 'grapefruits', 'apples', 'apricots', 'nectarines', 'oranges', 'peaches', 'pears', 'lemons']

Basket 2: ['grapes', 'dragonfruits', 'limes', 'papaya']

**Instructions 2/2**

**50 XP**

* [2](about:blank)
* Transfer fruits from basket1 to basket2 until the amount in basket2 becomes more or equal to the amount in basket1.

# Remove fruits from basket2 that are present in basket1

for item in basket1:

if item in basket2:

basket2.remove(item)

print('Basket 1: ' + str(basket1))

print('Basket 2: ' + str(basket2))

# Transfer fruits from basket1 to basket2

while len(basket1) != len(basket2):

item\_to\_transfer = basket1.pop()

basket2.append(item\_to\_transfer)

print('Basket 1: ' + str(basket1))

print('Basket 2: ' + str(basket2))

Well done! We practiced some of the list methods but there are many more! Don't hesitate to practice them as well.

# Operations on sets

Putting the information on sets in more mathematical terms, we can define the following operations given two sets XX and YY:

  X∩YX∩Y - the intersection between XX and YY (all elements which are in both XX and YY)

  X∪YX∪Y - the union between XX and YY (all elements which are either in XX or YY)

  X−YX−Y - the difference between XX and YY (all elements which are in XX but not in YY)

You are given 5 sets of integers A, B, C, D,E. What is the result of the following expression?

(A∪(B∩C))−(D∩E)

In [2]: (A.union((B.intersection(C))))-(D.intersection(E))

Out[2]: {1, 2, 3, 4, 5, 6, 7}

##### Possible Answers



{2}



{}



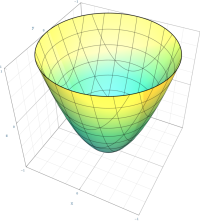
{1, 2}



{1, 2, 3, 4, 5, 6, 7} **(A)**

# Storing data in a dictionary

The surface you see below is called circular paraboloid:



It can be described by the following equation:

x2a2+y2a2=zx2a2+y2a2=z

Let's set the coefficient aa to 1. Therefore, the radius at each cut will be equal to z√z.

Your task is to create a dictionary that stores the mapping from the pair of coordinates (x,y)(x,y) to the zz coordinate (the lists storing considered ranges for xx and yy are given: range\_x and range\_y, respectively).

**Instructions 1/3**

**50 XP**

* [1](about:blank)
* [2](about:blank)
* [3](about:blank)
* Calculate the value for zz coordinate using coordinates xx and yy.
* Create a new key for the dictionary circ\_parab represented as a tuple containing xx and yy.
* Create a new key-value pair for circ\_parab.

range\_x = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]

range\_y = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]

In [1]: circ\_parab = dict()

for x in range\_x:

for y in range\_y:

# Calculate the value for z

z = x , y

# Create a new key for the dictionary

key = z

# Create a new key-value pair

circ\_parab[key] = z

**O/P :**

range\_x = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]

range\_y = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]

range\_x = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]

range\_y = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]

range\_x = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]

range\_y = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]

range\_x = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]

range\_y = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]

##### Instructions 2/3

**50 XP**

* [2](about:blank)
* [3](about:blank)

#### Question

What is the value of circ\_parab for the key (1.8, 1.4)?

##### Possible Answers



5.2 **(A)**



10.0



5.96



4.0

In [5]: circ\_parab[(1.8, 1.4)]

Out[5]: 5.2

##### Instructions 3/3

**0 XP**

* [3](about:blank)

#### Question

Is it possible to use a list instead of a tuple for a key in the circ\_parab dictionary?

##### Possible Answers



Yes, there is no substantial difference between two data structures in this regard.



No, because a list is mutable and the operation will result in TypeError. **(A)**



No, because a dictionary can accept only a tuple as a key.



No, because a list is an ordered sequence and the operation will result in TypeError.

**KEYS CAN REPRESENT ONLY IMMUTABLE OBJECTS**

# String indexing and concatenation

You are presented with one of the earliest known encryption techniques - Caesar cipher. It is based on a simple shift of each letter in a message by a certain number of positions down the given alphabet. For example, given the English alphabet, a shift of 1 for 'xyz' would imply 'yza' and *vice versa* in case of decryption. Notice that 'z' becomes 'a' in this case.

Thus, encryption/decryption requires two arguments: text and an integer key denoting the shift (key = 1 for the example above).

Your task is to create an encryption function given the English alphabet stored in the alphabet string.

##### Instructions 1/2

**50 XP**

* [1](about:blank)
* [2](about:blank)
* Fill in the blanks in the loop to create an encrypted text.
* Check the encryption function with the shift equals to 10 (it should return 'nkdkmkwz').

**def encrypt(text, key):**

**result = ''**

**# Fill in the blanks to create an encrypted text**

**for char in text.lower():**

**idx = (alphabet.index(char) + key) % len(alphabet)**

**result = result + alphabet[idx]**

**return result**

**# Check the encryption function with the shift equals to 10**

**print(encrypt("datacamp", 10))**

In [1]: alphabet

Out[1]: 'abcdefghijklmnopqrstuvwxyz'

<script.py> output:

Nkdkmkwz

##### Instructions 2/2

**50 XP**

* [2](about:blank)

#### Question

Great! Interestingly, decryption function is only different by the line you fixed in the for loop. What would be the corresponding change in the decrypt() function?

##### Possible Answers



idx = alphabet.index(char) – key **(A)**



idx = (alphabet.index(char) - key) % len(alphabet)



idx = alphabet.indx(char) + key



idx = (alphabet.indx(char) \* key) % len(alphabet)

Correct. It is enough to only subtract since Python allows negative indexing.

# Operations on strings

You are given the variable text storing the following string 'StRing ObJeCts haVe mANy inTEResting pROPerTies'.

Your task is to modify this string in such a way that would result in 'string OBJECTS have MANY interesting PROPERTIES' (every other word in text is uppercased and lowercased, otherwise). You will obtain this result in three steps.

##### Instructions 1/3

**35 XP**

* [1](about:blank)
* [2](about:blank)
* [3](about:blank)
* First, create a word list from the given string.

# Create a word list from the string stored in text

word\_list = text.split()

##### Instructions 2/3

**35 XP**

* [2](about:blank)
* [3](about:blank)
* Now, make every other word uppercased and lowercased, otherwise.

# Create a word list from the string stored in text

word\_list = text.split()

# Make every other word uppercased; otherwise - lowercased

for i in range(len(word\_list)):

if i%2 != 0:

word\_list[i] = word\_list[i].upper()

else:

word\_list[i] = word\_list[i].lower()

##### Instructions 3/3

**0 XP**

* [3](about:blank)
* Finally, join the words and form a new string and check the newly created string.

# Create a word list from the string stored in 'text'

word\_list = text.split()

# Make every other word uppercased; otherwise - lowercased

for i in range(len(word\_list)):

if (i + 1) % 2 == 0:

word\_list[i] = word\_list[i].upper()

else:

word\_list[i] = word\_list[i].lower()

# Join the words back and form a new string

new\_text = " ".join(word\_list)

print(new\_text)

Good! Having some exercises on strings is a good practice because coding interviews usually include string manipulation questions.

# Fixing string errors in a DataFrame

You are given the heroes dataset containing the information on different comic book heroes. However, you'll need to make some refinements in order to use this dataset further.

Comparing Eye color, Hair color, and Skin color columns, you can see that strings in the Hair color columns are capitalized, whereas in other two the strings are lowercased.

Moreover, some rows in the Gender column contain a spelling error (Fmale instead of Female).

Your task is to make the strings in the Hair column lowercased and to fix the spelling error in the Gender column.

##### Instructions 1/2

**50 XP**

* [1](about:blank)
* [2](about:blank)
* Make all the values in the Hair color column lowercased.

# Make all the values in the 'Hair color' column lowercased

heroes['Hair color'] = heroes['Hair color'].str.lower()

# Check the values in the 'Hair color' column

print(heroes['Hair color'].value\_counts())

##### Instructions 2/2

**50 XP**

* [2](about:blank)
* Substitute all the appearances of Fmale with Female in the Gender column.

# Make all the values in the 'Hair color' column lowercased

heroes['Hair color'] = heroes['Hair color'].str.lower()

# Check the values in the 'Hair color' column

print(heroes['Hair color'].value\_counts())

# Substitute 'Fmale' with 'Female' in the 'Gender' column

heroes['Gender'] = heroes['Gender'].str.replace('Fmale' , 'Female')

# Check if there is no occurences of 'Fmale'

print(heroes['Gender'].value\_counts())

Congratulations! You are ready to become a string armed ninja! Note that Series and DataFrames have their own .replace() method that deals with any kind of objects in addition to strings.

# Write a regular expression

Let's write some regular expressions!

Your task is to create a regular expression matching a valid temperature represented either in Celsius or Fahrenheit scale (*e.g.* '+23.5 C', '-4 F', '0.0 C') and to extract all the appearances from the given string text. Positive temperatures can be prefixed with + or contain no prefix (*e.g.* '5 F', '+5 F'). Negative temperatures must be prefixed with -. Zero temperature should not be prefixed with any symbol.

The re module is already imported.

Tip: the + symbol within the square brackets [] corresponds to the symbol itself (*e.g.* the regular expression [1a+] matches to '1', 'a', or '+').

##### Instructions 1/2

**50 XP**

* [1](about:blank)
* [2](about:blank)
* Define the pattern to search for valid temperatures

Let's consider the following temperatures using the Celsius scale: +23 C, 0 C, -20.0 C, -2.2 C, -5.65 C, 0.0001 C. To convert them to the Fahrenheit scale you have multiply the number by 9/5 and add 32 to the result. Therefore, the corresponding temperatures in the Fahrenheit scale will be: +73.4 F, 32 F, -4.0 F, +28.04 F, 21.83 F, +32.00018 F.

# Define the pattern to search for valid temperatures

pattern = re.compile('[+-]?\d+\.?\d\* [CF]')

# Print the temperatures out

print(re.findall(pattern, text))

##### Instructions 2/2

**50 XP**

* [2](about:blank)
* Create an object storing the matches using finditer().
* Loop over matches\_storage and print out item properties: the matching sequence, its start and end index.

# Define the pattern to search for valid temperatures

pattern = re.compile(r'[+-]?\d+\.?\d\* [CF]')

# Print the temperatures out

print(re.findall(pattern, text))

# Create an object storing the matches using 'finditer()'

matches\_storage = re.finditer(pattern , text)

# Loop over matches\_storage and print out item properties

for match in matches\_storage:

print('matching sequence = ' + match.group())

print('start index = ' + str(match.start()))

print('end index = ' + str(match.end()))

<script.py> output:

['+23 C', '0 C', '-20.0 C', '-2.2 C', '-5.65 C', '0.0001 C', '+73.4 F', '32 F', '-4.0 F', '+28.04 F', '21.83 F', '+32.00018 F']

matching sequence = +23 C

start index = 67

end index = 72

matching sequence = 0 C

start index = 74

end index = 77

matching sequence = -20.0 C

start index = 79

end index = 86

matching sequence = -2.2 C

start index = 88

end index = 94

matching sequence = -5.65 C

start index = 96

end index = 103

matching sequence = 0.0001 C

start index = 105

end index = 113

matching sequence = +73.4 F

start index = 292

end index = 299

matching sequence = 32 F

start index = 301

end index = 305

matching sequence = -4.0 F

start index = 307

end index = 313

matching sequence = +28.04 F

start index = 315

end index = 323

matching sequence = 21.83 F

start index = 325

end index = 332

matching sequence = +32.00018 F

start index = 334

Great job! We found all the valid temperatures and printed them out using the finditer() function!

# Find the incorrect pattern

Which of the following regular expressions will precisely match the long date format (for example, October 26, 1988 or Oct 26, 1988 with the first letter capitalized)?

Consider only the non-negative years.

The module re is already imported.

##### Instructions

**50 XP**

##### Possible Answers



\w+\s[1-3]?\d,\s\d+



[A-Z][a-z]+\s\d{1,2},\s\d+



[A-Z][a-z]+\s[1-3]?\d,\s\d+ **(A)**



[A-Z][a-z]+\s[1-3]?\d,\s\d\*

# Splitting by a pattern

You are given a text stored in the text variable.

Split the text in such a way that the resulting list has only words or numbers with no blank spaces or punctuation.

##### Instructions 1/2

**50 XP**

* [1](about:blank)
* [2](about:blank)
* Compile the regular expression.
* Split the text so that only words or numbers are included in the resulting list, and print the result.

# Compile the regular expression

pattern = re.compile(r'[,:\.\s]+')

# Split the text that only words or numbers are left

words = re.split(pattern, text)

print(words)

<script.py> output:

['Python', 'has', '4', 'main', 'data', 'structures', 'list', 'tuple', 'set', 'and', 'dictionary', '']

Python has 4 main data structures: list, tuple, set, and dictionary.

##### Instructions 2/2

**50 XP**

* [2](about:blank)
* Define a much easier way to extract words or numbers.

# Compile the regular expression

pattern = re.compile(r'[,:\.\s]+')

# Split the text that only words or numbers are left

words = re.split(pattern, text)

print(words)

# Define an easier way to extract words or numbers

**alt\_pattern = re.compile(r'\w')**

print(re.findall(alt\_pattern, text))

<script.py> output:

['Python', 'has', '4', 'main', 'data', 'structures', 'list', 'tuple', 'set', 'and', 'dictionary', '']

Python has 4 main data structures: list, tuple, set, and dictionary.

<script.py> output:

['Python', 'has', '4', 'main', 'data', 'structures', 'list', 'tuple', 'set', 'and', 'dictionary', '']

['P', 'y', 't', 'h', 'o', 'n', 'h', 'a', 's', '4', 'm', 'a', 'i', 'n', 'd', 'a', 't', 'a', 's', 't', 'r', 'u', 'c', 't', 'u', 'r', 'e', 's', 'l', 'i', 's', 't', 't', 'u', 'p', 'l', 'e', 's', 'e', 't', 'a', 'n', 'd', 'd', 'i', 'c', 't', 'i', 'o', 'n', 'a', 'r', 'y']

Good work! When given a task on regular expressions, there might be many ways to solve it. It is always better to think on the easiest one!

**enumerate()**

Let's enumerate! Your task is, given a string, to define the function retrieve\_character\_indices() that creates a dictionary character\_indices, where each key represents a unique character from the string and the corresponding value is a list containing the indices/positions of this letter in the string.

For example, passing the string 'ukulele' to the retrieve\_character\_indices() function should result in the following output: {'e': [4, 6], 'k': [1], 'l': [3, 5], 'u': [0, 2]}.

For this task, you are not allowed to use any string methods!

**Instructions**

**100 XP**

* Define the for loop that iterates over the characters in the string and their indices.
* Update the dictionary if the key already exists.
* Update the dictionary if the key is absent.

def retrieve\_character\_indices(string):

character\_indices = dict()

# Define the 'for' loop

for index, character in enumerate(string):

# Update the dictionary if the key already exists

if character in character\_indices:

character\_indices[character].append(index)

# Update the dictionary if the key is absent

else:

character\_indices[character] = [index]

return character\_indices

print(retrieve\_character\_indices('enumerate an Iterable'))

<script.py> output:

{'e': [0, 4, 8, 15, 20], 'n': [1, 11], 'u': [2], 'm': [3], 'r': [5, 16], 'a': [6, 10, 17], 't': [7, 14], ' ': [9, 12], 'I': [13], 'b': [18], 'l': [19]}

Very good! A little trick: actually, you can pass an integer value to the enumerate() initializer. In this case, it will start to count from that value.

# Iterators

Let's check your knowledge on Iterators!

As we discussed, all Iterables like list, set, or dict must have the associated Iterator. You are given the dictionary pets whose keys are Harry Potter characters and the values are the corresponding creature companions they had. Your task is to answer the set of questions regarding the Iterator created from the pets dictionary. Use the console to help you answer them! Pro tip: to break line in the console, use Shift + Enter.

##### Instructions 1/3

**35 XP**

* [1](about:blank)
* [2](about:blank)
* [3](about:blank)

#### Question

* What would be the second element of the Iterator created from the pets dictionary?

##### Possible Answers



'Harry'



'Hermione' **(A)**



'Hedwig the owl'



'Crookshanks the cat'

{'Harry': 'Hedwig the owl', 'Hermione': 'Crookshanks the cat', 'Ron': 'Scabbers the rat'}

##### Instructions 2/3

**35 XP**

* [2](about:blank)
* [3](about:blank)

#### Question

* Assuming that you retrieved the Iterator from the pets dictionary and called the next() function on it twice, what will be the output when you convert the Iterator to a list?

##### Possible Answers



['Ron'] **(A)**



[]



StopIteration error is raised



['Hermione', 'Ron']



['Harry', 'Hermione', 'Ron']

##### Instructions 3/3

**30 XP**

#### Question

* Assuming that you retrieved the Iterator from the pets dictionary and converted it to a list, what will be the output if you call the next() function on it?

##### Possible Answers



'Ron'



'Hermione'



'Harry'



StopIteration error is raised **(A)**

**Correct! The Iterator does not contain any more elements to go through after converting it to a list.**

# Traversing a DataFrame

Let's iterate through a DataFrame! You are given the heroes DataFrame you're already familiar with. This time, it contains only categorical data and no missing values. You have to create the following dictionary from this dataset:

* Each key is a column name.
* Each value is another dictionary:
  + Each key is a unique category from the column.
  + Each value is the amount of heroes falling into this category.

Tip: a Series object is also an Iterable. It traverses through the values it stores when you put it in a for loop or pass it to list(), tuple(), or set() initializers.

##### Instructions

**100 XP**

* Traverse through the columns in the heroes DataFrame.
* Retrieve the values stored in series in a list form.
* Traverse through unique categories in values.
* Count the appearance of category in values.

[**Take Hint (-30 XP)**](about:blank)

###### Incorrect Submission

Check the first for loop. Did you correctly specify the iterable part? Did you call set()?

Did you find this feedback helpful?

column\_counts = dict()

# Traverse through the columns in the heroes DataFrame

for column\_name, series in heroes.iteritems():

# Retrieve the values stored in series in a list form

values = list(series)

category\_counts = dict()

# Traverse through unique categories in values

for category in set(values):

# Count the appearance of category in values

category\_counts[category] = values.count(category)

column\_counts[column\_name] = category\_counts

print(column\_counts)

<script.py> output:

{'Gender': {'Female': 13, 'Male': 46}, 'Eye color': {'brown': 1, 'white': 8, 'blue': 11, 'green': 10, 'purple': 1, 'yellow': 5, 'yellow (without irises)': 1, 'red': 16, 'black': 4, 'grey': 1, 'gold': 1}, 'Race': {'Eternal': 1, 'Czarnian': 1, 'Inhuman': 1, 'Icthyo Sapien': 1, 'Human-Kree': 1, 'Human': 8, 'Alien': 4, 'Tamaranean': 1, 'Bizarro': 1, 'Android': 3, 'Human / Radiation': 3, 'Luphomoid': 1, "Yoda's species": 1, 'Neyaphem': 1, 'Martian': 1, 'Metahuman': 1, 'Human / Cosmic': 2, 'Mutant': 11, 'Ungaran': 1, 'Human / Altered': 1, 'Kakarantharaian': 1, 'Talokite': 1, 'Demon': 2, 'Frost Giant': 1, 'Korugaran': 1, 'Strontian': 1, 'Zen-Whoberian': 1, 'New God': 2, 'Bolovaxian': 1, 'God / Eternal': 3}, 'Hair color': {'Orange': 1, 'Black': 14, 'Purple': 1, 'Silver': 1, 'Blue': 2, 'Red / Orange': 1, 'Auburn': 1, 'Red': 2, 'No Hair': 25, 'Magenta': 1, 'Brown': 1, 'Blond': 2, 'Green': 3, 'White': 4}, 'Publisher': {'Marvel Comics': 32, 'IDW Publishing': 2, 'Dark Horse Comics': 1, 'George Lucas': 2, 'DC Comics': 22}, 'Skin color': {'white': 7, 'silver': 2, 'blue': 9, 'orange': 1, 'green': 17, 'pink': 2, 'purple': 3, 'yellow': 2, 'blue-white': 1, 'red': 8, 'gold': 2, 'grey': 4, 'gray': 1}, 'Alignment': {'bad': 21, 'good': 30, 'neutral': 8}}

Awesome! It is good to see that there are more good heroes than bad ones (according to the 'Alignment' key). However, the difference is not that big. Watch out!

## Iterating through multiple loops

numbers = [1, 2, 3]

letters = ['a', 'b', 'c']

pairs = [(i, j) **for** i **in** numbers **for** j **in** letters]

print(pairs)

[

(1, 'a'), (1, 'b'), (1, 'c'),

(2, 'a'), (2, 'b'), (2, 'c'),

(3, 'a'), (3, 'b'), (3, 'c'),

]

Play---

# Basic list comprehensions

For this task, you will have to create a bag-of-words representation of the spam email stored in the spam variable (you can explore the content using the shell). Recall that bag-of-words is simply a counter of unique words in a given text. This representation can be further used for text classification, *e.g.* for spam detection (given enough training examples).

We created a small auxiliary function create\_word\_list() to help you split a string into words, *e.g.* applying it to 'To infinity... and beyond!' will return ['To', 'infinity', 'and', 'beyond'].

##### Instructions 1/4

25 XP

* [1](about:blank)
* [2](about:blank)
* [3](about:blank)
* [4](about:blank)
* Convert the text to lower case and create a word list.

f\

Mute

Loaded: 0%

Progress: 0%

Remaining Time-1:15

Playback Rate

1.25x

Captions

Non-Fullscreen