**Exercise**

**Exercise**

**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](javascript:void(0))
* [2](javascript:void(0))
* Remove fruits from basket2 that are already present in basket1.

[**Take Hint (-15 XP)**](javascript:void(0))

Transfer fruits from basket1 to basket2 until the amount in basket2 becomes more or equal to the amount in basket1.

**Incorrect Submission**

Did you transfer fruits from basket1 to basket2? It looks like the contents of the variable basket1 are not correct.

**Hint**

* You have to compare lengths of the lists in the while loop.
* Recall the method that removes an item from a list and returns it at the same time.

# 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(basket2) < len(basket1):

item\_to\_transfer = basket1.pop()

basket2.append(item\_to\_transfer)

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']

<script.py> output:

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

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

Basket 1: ['banana', 'kiwifruits', 'grapefruits', 'apples', 'apricots', 'nectarines']

Basket 2: ['grapes', 'dragonfruits', 'limes', 'papaya', 'lemons', 'pears', 'peaches', 'oranges']

<script.py> output:

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

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

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

Basket 2: ['grapes', 'dragonfruits', 'limes', 'papaya', 'lemons', 'pears', 'peaches']

In [1]:

+85 XP

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)(A∪(B∩C))−(D∩E)

**Instructions**

**50 XP**

**Possible Answers**

* 

{2}

* 

{}

* 

{1, 2}

* 

{1, 2, 3, 4, 5, 6, 7}

**Submit Answer**

[**Take Hint (-15 XP)**](javascript:void(0))

A = {1, 2, 3, 4, 5, 6, 7}

B = {5, 7, 9, 11, 13, 15}

C = {1, 2, 8, 10, 11, 12, 13, 14, 15, 16, 17}

D = {1, 3, 5, 7, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}

E = {9, 10, 11, 12, 13, 14, 15}

In [1]: dir(set)

Out[1]:

['\_\_and\_\_',

'\_\_class\_\_',

'\_\_contains\_\_',

'\_\_delattr\_\_',

'\_\_dir\_\_',

'\_\_doc\_\_',

'\_\_eq\_\_',

'\_\_format\_\_',

'\_\_ge\_\_',

'\_\_getattribute\_\_',

'\_\_gt\_\_',

'\_\_hash\_\_',

'\_\_iand\_\_',

'\_\_init\_\_',

'\_\_init\_subclass\_\_',

'\_\_ior\_\_',

'\_\_isub\_\_',

'\_\_iter\_\_',

'\_\_ixor\_\_',

'\_\_le\_\_',

'\_\_len\_\_',

'\_\_lt\_\_',

'\_\_ne\_\_',

'\_\_new\_\_',

'\_\_or\_\_',

'\_\_rand\_\_',

'\_\_reduce\_\_',

'\_\_reduce\_ex\_\_',

'\_\_repr\_\_',

'\_\_ror\_\_',

'\_\_rsub\_\_',

'\_\_rxor\_\_',

'\_\_setattr\_\_',

'\_\_sizeof\_\_',

'\_\_str\_\_',

'\_\_sub\_\_',

'\_\_subclasshook\_\_',

'\_\_xor\_\_',

'add',

'clear',

'copy',

'difference',

'difference\_update',

'discard',

'intersection',

'intersection\_update',

'isdisjoint',

'issubset',

'issuperset',

'pop',

'remove',

'symmetric\_difference',

'symmetric\_difference\_update',

'union',

'update']

In [2]: B.intersection(C)

Out[2]: {11, 13, 15}

In [3]: A.union(B.intersection(C)).difference(D.intersection(E))

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

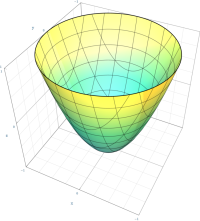
In [4]:

**Exercise**

**Exercise**

**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](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))

**Instructions 1/3**

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* 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.

[**Take Hint (-15 XP)**](javascript:void(0))

Instructions 2/3

50 XP

Question

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

Possible Answers

5.2

10.0

5.96

4.0

Instructions 3/3

0 XP

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.

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.

+100 XP

Correct. Keys can represent only immutable objects.

circ\_parab = dict()

for x in range\_x:

for y in range\_y:

# Calculate the value for z

z = x\*\*2 + y\*\*2

# Create a new key for the dictionary

key = (x, y)

# Create a new key-value pair

circ\_parab[key] = z

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]

In [1]: circ\_parab[1.8, 1.4]

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

circ\_parab[1.8, 1.4]

NameError: name 'circ\_parab' is not defined

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 [2]: circ\_parab[1.8, 1.4]

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

circ\_parab[1.8, 1.4]

NameError: name 'circ\_parab' is not defined

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 [3]: 1.8\*\*2+1.4\*\*2

Out[3]: 5.2

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]

In [4]:

**Exercise**

**Exercise**

**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](javascript:void(0))
* [2](javascript:void(0))
* 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').

[**Take Hint (-15 XP)**](javascript:void(0))

**Incorrect Submission**

Check the definition of encrypt(). Did you correctly specify the body? Are you sure you assigned the correct value to idx?

Have you used print(encrypt("datacamp", 10)) to do the appropriate printouts?

def encrypt(text, key):

encrypted\_text = ''

# Fill in the blanks to create an encrypted text

for char in text.lower():

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

encrypted\_text += alphabet[idx]

return encrypted\_text

# Check the encryption function with the shift equals to 10

print(encrypt("datacamp", 10))

**Instructions 2/2**

**50 XP**

* [2](javascript:void(0))

**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

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

idx = alphabet.indx(char) + key

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

**Incorrect Submission**

Incorrect. In case of negative indices % results in the same number.

+100 XP

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

In [1]: alphabet

Out[1]: 'abcdefghijklmnopqrstuvwxyz'

<script.py> output:

fcfcecgb

<script.py> output:

gdgdfdhc

<script.py> output:

ebebdbfa

<script.py> output:

hehegead

<script.py> output:

chchbhdg

<script.py> output:

ebebdbfa

<script.py> output:

fcfcecgb

<script.py> output:

hehegead

<script.py> output:

bgbgagcf

<script.py> output:

afafhfbe

<script.py> output:

dadacaeh

<script.py> output:

fcfcecgb

Traceback (most recent call last):

File "script.py", line 13, in <module>

print(encrypt("datacamp", 1))

File "script.py", line 8, in encrypt

encrypted\_text += alphabet[idx]

IndexError: string index out of range

File "script.py", line 7

idx = (alphabet\*2.index(char) + 10) #% len(text)

^

SyntaxError: invalid syntax

Traceback (most recent call last):

File "script.py", line 13, in <module>

print(encrypt("datacamp", 1))

File "script.py", line 8, in encrypt

encrypted\_text += alphabet[idx]

IndexError: string index out of range

<script.py> output:

ebebdbfa

<script.py> output:

fcfcecgb

Traceback (most recent call last):

File "script.py", line 13, in <module>

print(encrypt("datacamp", 10))

File "script.py", line 8, in encrypt

encrypted\_text += alphabet[alphabet.index(char) + key]

IndexError: string index out of range

Traceback (most recent call last):

File "script.py", line 13, in <module>

print(encrypt("datacamp", 10))

File "script.py", line 8, in encrypt

encrypted\_text += alphabet[alphabet.index(char) + key]

IndexError: string index out of range

<script.py> output:

nkdkmkwz

In [2]:

**Exercise**

**Exercise**

**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](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* First, create a word list from the given string.

[**Take Hint (-10 XP)**](javascript:void(0))

**Instructions 2/3**

**35 XP**

* [2](javascript:void(0))
* [3](javascript:void(0))
* Now, make every other word uppercased and lowercased, otherwise.

**Instructions 3/3**

**30 XP**

* [3](javascript:void(0))
* 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[wl].lower() if wl%2==0 else word\_list[wl].upper() for wl in range(len(word\_list))])

print(new\_text)

StRing ObJeCts haVe mANy inTEResting pROPerTies

StRing ObJeCts haVe mANy inTEResting pROPerTies

StRing ObJeCts haVe mANy inTEResting pROPerTies

StRing ObJeCts haVe mANy inTEResting pROPerTies

StRing ObJeCts haVe mANy inTEResting pROPerTies

<script.py> output:

string OBJECTS have MANY interesting PROPERTIES

In [1]:

+100 XP

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

**Exercise**

**Exercise**

**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](javascript:void(0))
* [2](javascript:void(0))
* Make all the values in the Hair color column lowercased.

[**Take Hint (-15 XP)**](javascript:void(0))

* Substitute all the appearances of Fmale with Female in the Gender column.

[**Take Hint (-15 XP)**](javascript:void(0))

# 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())

In [1]: heroes

Out[1]:

name Gender Eye color Race Hair color Height Publisher Skin color Alignment Weight

0 A-Bomb Male yellow Human No Hair 203.0 Marvel Comics NaN good 441.0

1 Abe Sapien Male blue Icthyo Sapien No Hair 191.0 Dark Horse Comics blue good 65.0

2 Abin Sur Male blue Ungaran No Hair 185.0 DC Comics red good 90.0

3 Abomination Male green Human / Radiation No Hair 203.0 Marvel Comics NaN bad 441.0

4 Abraxas Male blue Cosmic Entity Black NaN Marvel Comics NaN bad NaN

5 Absorbing Man Male blue Human No Hair 193.0 Marvel Comics NaN bad 122.0

6 Adam Monroe Male blue NaN Blond NaN NBC - Heroes NaN good NaN

7 Adam Strange Male blue Human Blond 185.0 DC Comics NaN good 88.0

8 Agent 13 Fmale blue NaN Blond 173.0 Marvel Comics NaN good 61.0

9 Agent Bob Male brown Human Brown 178.0 Marvel Comics NaN good 81.0

10 Agent Zero Male NaN NaN NaN 191.0 Marvel Comics NaN good 104.0

11 Air-Walker Male blue NaN White 188.0 Marvel Comics NaN bad 108.0

12 Ajax Male brown Cyborg Black 193.0 Marvel Comics NaN bad 90.0

13 Alan Scott Male blue NaN Blond 180.0 DC Comics NaN good 90.0

14 Alex Mercer Male NaN Human NaN NaN Wildstorm NaN bad NaN

15 Alex Woolsly Male NaN NaN NaN NaN NBC - Heroes NaN good NaN

16 Alfred Pennyworth Male blue Human Black 178.0 DC Comics NaN good 72.0

17 Alien Male NaN Xenomorph XX121 No Hair 244.0 Dark Horse Comics black bad 169.0

18 Allan Quatermain Male NaN NaN NaN NaN Wildstorm NaN good NaN

19 Amazo Male red Android NaN 257.0 DC Comics NaN bad 173.0

20 Ammo Male brown Human Black 188.0 Marvel Comics NaN bad 101.0

21 Ando Masahashi Male NaN NaN NaN NaN NBC - Heroes NaN good NaN

22 Angel Male blue NaN Blond 183.0 Marvel Comics NaN good 68.0

23 Angel Male NaN Vampire NaN NaN Dark Horse Comics NaN good NaN

24 Angel Dust Fmale yellow Mutant Black 165.0 Marvel Comics NaN good 57.0

25 Angel Salvadore Female brown NaN Black 163.0 Marvel Comics NaN good 54.0

26 Angela Female NaN NaN NaN NaN Image Comics NaN bad NaN

27 Animal Man Male blue Human Blond 183.0 DC Comics NaN good 83.0

28 Annihilus Male green NaN No Hair 180.0 Marvel Comics NaN bad 90.0

29 Ant-Man Male blue Human Blond 211.0 Marvel Comics NaN good 122.0

.. ... ... ... ... ... ... ... ... ... ...

704 Walrus Male blue Human Black 183.0 Marvel Comics NaN bad 162.0

705 War Machine Male brown Human Brown 185.0 Marvel Comics NaN good 95.0

706 Warbird Female blue NaN Blond 180.0 Marvel Comics NaN good 54.0

707 Warlock Male red NaN Blond 188.0 Marvel Comics NaN good 108.0

708 Warp Male brown NaN Black 173.0 DC Comics NaN bad 67.0

709 Warpath Male brown Mutant Black 218.0 Marvel Comics NaN good 158.0

710 Wasp Female blue Human Auburn 163.0 Marvel Comics NaN good 50.0

711 Watcher Male NaN NaN NaN NaN Marvel Comics NaN good NaN

712 Weapon XI Male NaN NaN NaN NaN Marvel Comics NaN bad NaN

713 White Canary Female brown Human Black NaN DC Comics NaN bad NaN

714 White Queen Female blue NaN Blond 178.0 Marvel Comics NaN good 65.0

715 Wildfire Male NaN NaN NaN NaN DC Comics NaN good NaN

716 Winter Soldier Male brown Human Brown 175.0 Marvel Comics NaN good 117.0

717 Wiz Kid NaN brown NaN Black 140.0 Marvel Comics NaN good 39.0

718 Wolfsbane Female green NaN Auburn 366.0 Marvel Comics NaN good 473.0

719 Wolverine Male blue Mutant Black 160.0 Marvel Comics NaN good 135.0

720 Wonder Girl Fmale blue Demi-God Blond 165.0 DC Comics NaN good 51.0

721 Wonder Man Male red NaN Black 188.0 Marvel Comics NaN good 171.0

722 Wonder Woman Female blue Amazon Black 183.0 DC Comics NaN good 74.0

723 Wondra Female NaN NaN NaN NaN Marvel Comics NaN good NaN

724 Wyatt Wingfoot Male brown NaN Black 196.0 Marvel Comics NaN good 117.0

725 X-23 Female green Mutant / Clone Black 155.0 Marvel Comics NaN good 50.0

726 X-Man Male blue NaN Brown 175.0 Marvel Comics NaN good 61.0

727 Yellow Claw Male blue NaN No Hair 188.0 Marvel Comics NaN bad 95.0

728 Yellowjacket Male blue Human Blond 183.0 Marvel Comics NaN good 83.0

729 Yellowjacket II Female blue Human Strawberry Blond 165.0 Marvel Comics NaN good 52.0

730 Ymir Male white Frost Giant No Hair 304.8 Marvel Comics white good NaN

731 Yoda Male brown Yoda's species White 66.0 George Lucas green good 17.0

732 Zatanna Fmale blue Human Black 170.0 DC Comics NaN good 57.0

733 Zoom Male red NaN Brown 185.0 DC Comics NaN bad 81.0

[734 rows x 10 columns]

<script.py> output:

black 161

blond 102

brown 87

no hair 75

red 51

white 23

auburn 13

green 8

strawberry blond 7

grey 5

purple 5

brown / white 4

silver 4

blue 3

yellow 2

orange 2

orange / white 1

pink 1

brown / black 1

gold 1

black / blue 1

red / grey 1

red / white 1

indigo 1

red / orange 1

magenta 1

Name: Hair color, dtype: int64

<script.py> output:

black 161

blond 102

brown 87

no hair 75

red 51

white 23

auburn 13

green 8

strawberry blond 7

grey 5

purple 5

brown / white 4

silver 4

blue 3

yellow 2

orange 2

orange / white 1

pink 1

brown / black 1

gold 1

black / blue 1

red / grey 1

red / white 1

indigo 1

red / orange 1

magenta 1

Name: Hair color, dtype: int64

Male 505

Female 200

Name: Gender, dtype: int64

In [2]:

+100 XP

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.

**Exercise**

**Exercise**

**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', '73.45 F') and to extract all the appearances from the given string text. Positive temperatures can be with or without the + prefix (*e.g.* '5 F', '+5 F'). Negative temperatures must be prefixed with -. Zero temperature can be used with a prefix or without.

The re module is already imported.

Tips:

* The + symbol within the square brackets [] matches the + symbol itself (*e.g.* the regular expression [1a+] matches to '1', 'a', or '+').
* You can also apply ? to the characters within the square brackets [] to make the set optional (*e.g.* [ab]? matches to 'a', 'b', or '').

**Instructions 1/2**

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))

**Instructions 1/2**

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Define a pattern to search for valid temperatures in text.

[**Take Hint (-15 XP)**](javascript:void(0))

**Hint**

* Remember to use the r prefix before defining a regular expression within the quotation marks ''.
* There are several occasions where square brackets [] might be useful.
* Remember that the dot character prefixed with a backslash matches to the dot character.

**Incorrect Submission**

The print() statement didn't produce the expected output. Is your pattern to search for valid temperatures correct?

# Define a pattern to search for valid temperatures in text

pattern = re.compile(r'[0-9.+-]+ F|[0-9.+-]+ C')

# Print the temperatures out

print(re.findall(pattern, text))

**Instructions 2/2**

**50 XP**

* [2](javascript:void(0))
* 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.

[**Take Hint (-15 XP)**](javascript:void(0))

# 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()))

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 to 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.

Traceback (most recent call last):

File "script.py", line 2, in <module>

pattern = re.compile(r'\* F|\* C')

File "script.py", line 233, in compile

return \_compile(pattern, flags)

File "script.py", line 301, in \_compile

p = sre\_compile.compile(pattern, flags)

File "script.py", line 562, in compile

p = sre\_parse.parse(p, flags)

File "script.py", line 855, in parse

p = \_parse\_sub(source, pattern, flags & SRE\_FLAG\_VERBOSE, 0)

File "script.py", line 416, in \_parse\_sub

not nested and not items))

File "script.py", line 616, in \_parse

source.tell() - here + len(this))

sre\_constants.error: nothing to repeat at position 0

Traceback (most recent call last):

File "script.py", line 2, in <module>

pattern = re.compile(r'\*F|\*C')

File "script.py", line 233, in compile

return \_compile(pattern, flags)

File "script.py", line 301, in \_compile

p = sre\_compile.compile(pattern, flags)

File "script.py", line 562, in compile

p = sre\_parse.parse(p, flags)

File "script.py", line 855, in parse

p = \_parse\_sub(source, pattern, flags & SRE\_FLAG\_VERBOSE, 0)

File "script.py", line 416, in \_parse\_sub

not nested and not items))

File "script.py", line 616, in \_parse

source.tell() - here + len(this))

sre\_constants.error: nothing to repeat at position 0

Traceback (most recent call last):

File "script.py", line 2, in <module>

pattern = re.compile(r'\*F | \*C')

File "script.py", line 233, in compile

return \_compile(pattern, flags)

File "script.py", line 301, in \_compile

p = sre\_compile.compile(pattern, flags)

File "script.py", line 562, in compile

p = sre\_parse.parse(p, flags)

File "script.py", line 855, in parse

p = \_parse\_sub(source, pattern, flags & SRE\_FLAG\_VERBOSE, 0)

File "script.py", line 416, in \_parse\_sub

not nested and not items))

File "script.py", line 616, in \_parse

source.tell() - here + len(this))

sre\_constants.error: nothing to repeat at position 0

Traceback (most recent call last):

File "script.py", line 2, in <module>

pattern = re.compile(r'(\*F)|(\*C)')

File "script.py", line 233, in compile

return \_compile(pattern, flags)

File "script.py", line 301, in \_compile

p = sre\_compile.compile(pattern, flags)

File "script.py", line 562, in compile

p = sre\_parse.parse(p, flags)

File "script.py", line 855, in parse

p = \_parse\_sub(source, pattern, flags & SRE\_FLAG\_VERBOSE, 0)

File "script.py", line 416, in \_parse\_sub

not nested and not items))

File "script.py", line 765, in \_parse

p = \_parse\_sub(source, state, sub\_verbose, nested + 1)

File "script.py", line 416, in \_parse\_sub

not nested and not items))

File "script.py", line 616, in \_parse

source.tell() - here + len(this))

sre\_constants.error: nothing to repeat at position 1

<script.py> output:

[]

<script.py> output:

[' C', ' C', ' C', ' C', ' C', ' C', ' C', ' F', ' F', ' F', ' F', ' F', ' F', ' F', ' F']

Traceback (most recent call last):

File "script.py", line 2, in <module>

pattern = re.compile(r'\*.F|\*.C')

File "script.py", line 233, in compile

return \_compile(pattern, flags)

File "script.py", line 301, in \_compile

p = sre\_compile.compile(pattern, flags)

File "script.py", line 562, in compile

p = sre\_parse.parse(p, flags)

File "script.py", line 855, in parse

p = \_parse\_sub(source, pattern, flags & SRE\_FLAG\_VERBOSE, 0)

File "script.py", line 416, in \_parse\_sub

not nested and not items))

File "script.py", line 616, in \_parse

source.tell() - here + len(this))

sre\_constants.error: nothing to repeat at position 0

<script.py> output:

['e C', '3 C', '0 C', '0 C', '2 C', '5 C', '1 C', 'e F', 'e F', '4 F', '2 F', '0 F', '4 F', '3 F', '8 F']

<script.py> output:

['the C', '+23 C', ', 0 C', '0.0 C', '2.2 C', '.65 C', '001 C', 'the F', 'the F', '3.4 F', ' 32 F', '4.0 F', '.04 F', '.83 F', '018 F']

In [1]: text

Out[1]: "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 to 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."

<script.py> output:

[]

<script.py> output:

['3 C', '0 C', '0 C', '2 C', '5 C', '1 C', '4 F', '2 F', '0 F', '4 F', '3 F', '8 F']

<script.py> output:

['3 C', '0 C', '0 C', '2 C', '5 C', '1 C', '3.4 F', '4.0 F', '018 F']

<script.py> output:

['0.0 C', '2.2 C', '001 C', '3.4 F', '4.0 F', '018 F']

<script.py> output:

['23 C', '0 C', '0 C', '2 C', '65 C', '0001 C', '4 F', '32 F', '0 F', '04 F', '83 F', '00018 F']

<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']

In [2]: text

Out[2]: "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 to 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."

<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']

<script.py> output:

['+23 C', '0 C', '20.0 C', '2.2 C', '5.65 C', '0.0001 C']

<script.py> output:

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

<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']

<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']

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 to 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.

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 to 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.

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 to 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.

In [3]: re.finditer(pattern, text)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

re.finditer(pattern, text)

NameError: name 'pattern' is not defined

In [4]: re.finditer(pattern, text)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

re.finditer(pattern, text)

NameError: name 'pattern' is not defined

In [5]: re.finditer(pattern, text)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

re.finditer(pattern, text)

NameError: name 'pattern' is not defined

In [6]: # 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 = ' + \_\_\_\_)

print('start index = ' + str(\_\_\_\_))

print('end index = ' + str(\_\_\_\_))

['+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']

Traceback (most recent call last):

File "<stdin>", line 12, in <module>

print('matching sequence = ' + \_\_\_\_)

NameError: name '\_\_\_\_' is not defined

In [7]: re.finditer(pattern, text)

Out[7]: <callable\_iterator at 0x7fd5a1bcdda0>

In [8]: list(re.finditer(pattern, text))

Out[8]:

[<\_sre.SRE\_Match object; span=(67, 72), match='+23 C'>,

<\_sre.SRE\_Match object; span=(74, 77), match='0 C'>,

<\_sre.SRE\_Match object; span=(79, 86), match='-20.0 C'>,

<\_sre.SRE\_Match object; span=(88, 94), match='-2.2 C'>,

<\_sre.SRE\_Match object; span=(96, 103), match='-5.65 C'>,

<\_sre.SRE\_Match object; span=(105, 113), match='0.0001 C'>,

<\_sre.SRE\_Match object; span=(295, 302), match='+73.4 F'>,

<\_sre.SRE\_Match object; span=(304, 308), match='32 F'>,

<\_sre.SRE\_Match object; span=(310, 316), match='-4.0 F'>,

<\_sre.SRE\_Match object; span=(318, 326), match='+28.04 F'>,

<\_sre.SRE\_Match object; span=(328, 335), match='21.83 F'>,

<\_sre.SRE\_Match object; span=(337, 348), match='+32.00018 F'>]

<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 = 295

end index = 302

matching sequence = 32 F

start index = 304

end index = 308

matching sequence = -4.0 F

start index = 310

end index = 316

matching sequence = +28.04 F

start index = 318

end index = 326

matching sequence = 21.83 F

start index = 328

end index = 335

matching sequence = +32.00018 F

start index = 337

end index = 348

In [9]:

+85 XP

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

**Find the correct pattern**

You visit a website and it asks you to fill in a registration form. The username section requires you to choose a name between 3 and 16 characters long. It can only include alphanumeric characters (no capital letters), hyphens, and underscores. Which of the following patterns will your input be matched against?

The symbols ^ and $ were not discussed in the video lecture: they simply indicate the beginning and end of a line (in this case, the beginning and end of the chosen username).

The module re is already imported for you.

**Instructions**

**50 XP**

**Instructions**

**50 XP**

**Possible Answers**

* 

^[a-zA-Z0-9\_-]{3,16}$

* 

^[a-z0-9\_-]{3,17}$

* 

^[a-z0-9\_-]{3,16}$

* 

^[a-z0-9\_]{3,16}$

**Submit Answer**

[**Take Hint (-15 XP)**](javascript:void(0))

+50 XP

Correct. This is the pattern your input will be matched against.

**Splitting by a pattern**

You are given the movies list where each element stores a movie name, its release date, and the director (*e.g.* "The Godfather, 1972, Francis Ford Coppola").

Let's practice some splitting with the help of regular expressions. Your task is to retrieve from each element of the list its name and the director. For example, if the element is "The Godfather, 1972, Francis Ford Coppola", the result would be:

['The Godfather', 'Francis Ford Coppola']

Eventually, this result should be modified to represent a single string, *e.g.*

"The Godfather, Francis Ford Coppola"

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Compile a regular expression that splits strings in movies into a movie name and its director.
* Retrieve a movie name and its director using re.split().
* Create a new string with a movie name and its director separated by ,.
* Append the resulting string to movies\_without\_year.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you call re.compile()?

Check your call of re.compile(). Did you correctly specify the first argument? Did you use r in front of your regular expression string to indicate it's a regular expression?

Check your call of re.compile(). Did you correctly specify the first argument? Did you use a comma followed by a space twice in your regular expression?

Check your call of re.compile(). Did you correctly specify the first argument? Did you use \d to refer to a digit in your regular expression?

Check your call of re.compile(). Did you correctly specify the first argument? Did you use \d+ to refer to at least one digit in your regular expression?

Your regular expression does not match the expected expression.

# Compile a regular expression

pattern = re.compile(r', \d+')

movies\_without\_year = []

def isnum(text):

try: int(text); return 1

except ValueError: return 0

for movie in movies:

# Retrieve a movie name and its director

#split\_result = re.split(\_\_\_\_)

split\_result = movie.split(', ')

# Create a new string with a movie name and its director

split\_result.pop([sr for sr in range(len(split\_result)) if isnum(split\_result[sr])][-1])

movie\_without\_year = ', '.join(split\_result)

# Append the resulting string to movies\_without\_year

movies\_without\_year += [movie\_without\_year]

for movie in movies\_without\_year:

print(movie)

+100 XP

Very good! Figuring out the right regular expression is often a challenging task. But now you can cope with them!

1984, 1984, Michael Radford

The Good, the Bad and the Ugly, 1966, Sergio Leone

Terminator 2: Judgment Day, 1991, James Cameron

Harry Potter and the Philosopher's Stone, 2001, Chris Columbus

Back to the Future, 1985, Robert Zemeckis

No Country for Old Men, 2007, Joel Coen, Ethan Coen

In [1]: pattern

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

pattern

NameError: name 'pattern' is not defined

In [2]: movies

Out[2]:

['1984, 1984, Michael Radford',

'The Good, the Bad and the Ugly, 1966, Sergio Leone',

'Terminator 2: Judgment Day, 1991, James Cameron',

"Harry Potter and the Philosopher's Stone, 2001, Chris Columbus",

'Back to the Future, 1985, Robert Zemeckis',

'No Country for Old Men, 2007, Joel Coen, Ethan Coen']

<script.py> output:

1984, Michael Radford

The Good, 1966, Sergio Leone

Terminator 2: Judgment Day, James Cameron

Harry Potter and the Philosopher's Stone, Chris Columbus

Back to the Future, Robert Zemeckis

No Country for Old Men, Joel Coen, Ethan Coen

<script.py> output:

1984, Michael Radford

The Good, 1966, Sergio Leone

Terminator 2: Judgment Day, James Cameron

Harry Potter and the Philosopher's Stone, Chris Columbus

Back to the Future, Robert Zemeckis

No Country for Old Men, Joel Coen, Ethan Coen

<script.py> output:

1984, Michael Radford

The Good, 1966, Sergio Leone

Terminator 2: Judgment Day, James Cameron

Harry Potter and the Philosopher's Stone, Chris Columbus

Back to the Future, Robert Zemeckis

No Country for Old Men, Joel Coen, Ethan Coen

<script.py> output:

1984, Michael Radford

The Good, 1966, Sergio Leone

Terminator 2: Judgment Day, James Cameron

Harry Potter and the Philosopher's Stone, Chris Columbus

Back to the Future, Robert Zemeckis

No Country for Old Men, Joel Coen, Ethan Coen

<script.py> output:

1984, Michael Radford

The Good, 1966, Sergio Leone

Terminator 2: Judgment Day, James Cameron

Harry Potter and the Philosopher's Stone, Chris Columbus

Back to the Future, Robert Zemeckis

No Country for Old Men, Joel Coen, Ethan Coen

<script.py> output:

1984, Michael Radford

The Good, 1966, Sergio Leone

Terminator 2: Judgment Day, James Cameron

Harry Potter and the Philosopher's Stone, Chris Columbus

Back to the Future, Robert Zemeckis

No Country for Old Men, Joel Coen, Ethan Coen

<script.py> output:

1984, 1984

The Good, the Bad and the Ugly, 1966

Terminator 2: Judgment Day, 1991

Harry Potter and the Philosopher's Stone, 2001

Back to the Future, 1985

No Country for Old Men, 2007, Joel Coen

<script.py> output:

1984, Michael Radford

The Good, the Bad and the Ugly, Sergio Leone

Terminator 2: Judgment Day, James Cameron

Harry Potter and the Philosopher's Stone, Chris Columbus

Back to the Future, Robert Zemeckis

No Country for Old Men, 2007, Ethan Coen

Traceback (most recent call last):

File "script.py", line 16, in <module>

[split\_result.pop(sr) for sr in range(len(split\_result)) if isnum(split\_result[sr])]

File "script.py", line 16, in <listcomp>

[split\_result.pop(sr) for sr in range(len(split\_result)) if isnum(split\_result[sr])]

IndexError: list index out of range

<script.py> output:

[0, 1]

[2]

[1]

[1]

[1]

[1]

1984, 1984, Michael Radford

The Good, the Bad and the Ugly, 1966, Sergio Leone

Terminator 2: Judgment Day, 1991, James Cameron

Harry Potter and the Philosopher's Stone, 2001, Chris Columbus

Back to the Future, 1985, Robert Zemeckis

No Country for Old Men, 2007, Joel Coen, Ethan Coen

<script.py> output:

1

2

1

1

1

1

1984, 1984, Michael Radford

The Good, the Bad and the Ugly, 1966, Sergio Leone

Terminator 2: Judgment Day, 1991, James Cameron

Harry Potter and the Philosopher's Stone, 2001, Chris Columbus

Back to the Future, 1985, Robert Zemeckis

No Country for Old Men, 2007, Joel Coen, Ethan Coen

<script.py> output:

1984, Michael Radford

The Good, the Bad and the Ugly, Sergio Leone

Terminator 2: Judgment Day, James Cameron

Harry Potter and the Philosopher's Stone, Chris Columbus

Back to the Future, Robert Zemeckis

No Country for Old Men, Joel Coen, Ethan Coen

In [3]:

**Exercise**

**Exercise**

**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.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Check the definition of retrieve\_character\_indices(). Did you correctly specify the body? Running it generated an error: name '\_\_\_\_' is not defined.

Check the definition of retrieve\_character\_indices(). Did you correctly specify the body? Are you sure you assigned the correct value to character\_indices?

Check the definition of retrieve\_character\_indices(). Did you correctly specify the body? Running it generated an error: not enough values to unpack (expected 2, got 1).

def retrieve\_character\_indices(string):

character\_indices = dict()

# Define the 'for' loop

for index, character in list(character\_indices):

# Update the dictionary if the key already exists

if character in character\_indices:

character\_indices[character].update([index])

# Update the dictionary if the key is absent

else:

character\_indices[character] = [index]

return character\_indices

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

**Hint**

* Recall how to check if a key is present in a dictionary.
* Recall how to add a new item to a list.

def retrieve\_character\_indices(string):

character\_indices = dict()

# Define the 'for' loop

for index, character in zip(list(string)):

# Update the dictionary if the key already exists

if character in character\_indices:

character\_indices[character].update([index])

# Update the dictionary if the key is absent

else:

character\_indices[character] = [index]

return character\_indices

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

solution.py

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'))

In [1]: def retrieve\_character\_indices(string):

character\_indices = dict()

# Define the 'for' loop

for index, character in list(string):

# Update the dictionary if the key already exists

if character in character\_indices:

character\_indices[character].update([index])

# Update the dictionary if the key is absent

else:

character\_indices[character] = [index]

return character\_indices

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

Traceback (most recent call last):

File "<stdin>", line 14, in <module>

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

File "<stdin>", line 4, in retrieve\_character\_indices

for index, character in list(string):

ValueError: not enough values to unpack (expected 2, got 1)

Traceback (most recent call last):

File "script.py", line 14, in <module>

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

File "script.py", line 4, in retrieve\_character\_indices

for index, character in zip(string):

ValueError: not enough values to unpack (expected 2, got 1)

Traceback (most recent call last):

File "script.py", line 14, in <module>

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

File "script.py", line 4, in retrieve\_character\_indices

for index, character in zip(list(string)):

ValueError: not enough values to unpack (expected 2, got 1)

In [2]:

+0 XP

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.

**Exercise**

**Exercise**

**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 a line in the IPython Shell (**not** the script.py section), use Shift + Enter.

**Instructions 1/3**

**35 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))

**Question**

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

**Possible Answers**

'Harry'

'Hermione'

'Hedwig the owl'

'Crookshanks the cat'

Instructions 2/3

35 XP

2

3

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']

[]

StopIteration error is raised

['Hermione', 'Ron']

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

**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

+100 XP

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)**](javascript:void(0))

column\_counts = dict()

# Traverse through the columns in the heroes DataFrame

for column\_name, series in heroes:

# 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)

**Incorrect Submission**

Check the first for loop. Did you correctly specify the iterable part? Expected heroes.iteritems(), but got heroes.

Gender Eye color Race Hair color Publisher Skin color Alignment

name

Abe Sapien Male blue Icthyo Sapien No Hair Dark Horse Comics blue good

Abin Sur Male blue Ungaran No Hair DC Comics red good

Apocalypse Male red Mutant Black Marvel Comics grey bad

Archangel Male blue Mutant Blond Marvel Comics blue good

Ardina Female white Alien Orange Marvel Comics gold good

Traceback (most recent call last):

File "script.py", line 4, in <module>

for column\_name, series in heroes:

ValueError: too many values to unpack (expected 2)

<script.py> output:

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

In [1]:

+100 XP

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!

**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](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* [4](javascript:void(0))
* Convert the text to lower case and create a word list.

[**Take Hint (-7 XP)**](javascript:void(0))

**Incorrect Submission**

Did you call create\_word\_list()?

Create a set that will store only unique words from the list.

Using list comprehension, create a dictionary that counts a word appearance in the word list.

**Incorrect Submission**

Check the first list comprehension. Did you correctly specify the iterable part? Expected something different.

Check the first list comprehension. Did you correctly specify the body? Running it generated an error: 'set' object has no attribute 'count'.

Print words that appear in the word\_counter more than once.

**Incorrect Submission**

Check the first for loop. Did you correctly specify the iterable part? Expected something different.

# Convert the text to lower case and create a word list

words = create\_word\_list(spam.lower())

# Create a set storing only unique words

word\_set = set(words)

# Create a dictionary that counts each word in the list

tuples = [(word, words.count(word)) for word in word\_set]

word\_counter = dict(tuples)

# Printing words that appear more than once

for (key, value) in word\_counter.items():

if value > 1:

print("{}: {}".format(key, value))

Dear User,

Our Administration Team needs to inform you that you are reaching the storage limit of your Mailbox account.

You have to verify your account within the next 24 hours.

Otherwise, it will not be possible to use the service.

Please, click on the link below to verify your account and continue using our service.

Your Administration Team.

Traceback (most recent call last):

File "script.py", line 8, in <module>

tuples = [(word, word\_set.count) for word in word\_set]

File "script.py", line 8, in <listcomp>

tuples = [(word, word\_set.count) for word in word\_set]

AttributeError: 'set' object has no attribute 'count'

Dear User,

Our Administration Team needs to inform you that you are reaching the storage limit of your Mailbox account.

You have to verify your account within the next 24 hours.

Otherwise, it will not be possible to use the service.

Please, click on the link below to verify your account and continue using our service.

Your Administration Team.

Traceback (most recent call last):

File "script.py", line 13, in <module>

if value > 1:

TypeError: '>' not supported between instances of 'str' and 'int'

<script.py> output:

administration: 2

verify: 2

service: 2

your: 4

you: 3

account: 3

our: 2

the: 4

to: 4

team: 2

In [1]:

+100 XP

Great! You can observe that one of the most frequent words in the email are account, service, administration, and verify. This might be a security email making sure an unusual login attempt was made by the owner of the account... Or it can be a phishing email!

**Prime number sequence**

A prime number is a positive number that is divisible only by 1 or itself (*e.g.* 3, 7, 11 *etc.*). However, 1 is not a prime number.

Your task is, given a list of candidate numbers cands, to filter only prime numbers in a new list primes.

But first, you need to create a function is\_prime() that returns True if the input number nn is prime or False, otherwise. A number is prime if it is not divisible by any integer number from 2 to n−−√n (any number nn is not divisible by anything higher than n−−√n).

Tip: you might need to use the % operator that calculates a remainder from a division (*e.g.* 8 % 3 is 2).

The math module is already imported.

**Instructions 1/2**

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))

**Instructions 1/2**

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Define the initial check: numbers lower than 2 are not prime.
* Define the loop checking if the number n is not prime.

[**Take Hint (-15 XP)**](javascript:void(0))

**Incorrect Submission**

Check the first if statement. Did you correctly specify the condition? Running it generated an error: name 'test\_prime\_list' is not defined.

Your code could not be parsed due to an error in the indentation:  
inconsistent use of tabs and spaces in indentation (script.py, line 3).

Check the first for loop. Did you correctly specify the iterable part? Expected range(2, 2), but got range(2, 3).

Check the first for loop. Did you correctly specify the iterable part? Expected range(2, 2), but got range(2, 4).

* Filter prime numbers from cands into the primes list.

[**Take Hint (-15 XP)**](javascript:void(0))

def is\_prime(n):

# Define the initial check

if n < 2:

return False

# Define the loop checking if a number is not prime

for i in range(2, int(math.sqrt(n)) + 1):

if n % i == 0:

return False

return True

# Filter prime numbers into the new list

primes = [num for num in cands if is\_prime(num)]

print("primes = " + str(primes))

cands = [1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49]

In [1]: def is\_prime(num,print\_tracker=False):

global test\_prime\_list

test\_prime\_list = []

for i in range(2,num):

division = num/i

if str(division)[-2:] != '.0': test\_prime\_list.append(type(division) != type(1))

if print\_tracker: print(num,i,division,type(division) != type(1),sep='\t'\*2)

if test\_prime\_list == [True]\*(num-2): return True

else: return False

File "script.py", line 3

for i in range(2,num):

^

TabError: inconsistent use of tabs and spaces in indentation

File "script.py", line 3

for i in range(2,num):

^

TabError: inconsistent use of tabs and spaces in indentation

File "script.py", line 3

for i in range(2,num):

^

TabError: inconsistent use of tabs and spaces in indentation

In [2]: def is\_prime(num,print\_tracker=False):

test\_prime\_list = []

for i in range(2,num):

division = num/i

if str(division)[-2:] != '.0': test\_prime\_list.append(type(division) != type(1))

if print\_tracker: print(num,i,division,type(division) != type(1),sep='\t'\*2)

if test\_prime\_list == [True]\*(num-2): return True

else: return False

cands = [1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49]

cands = [1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49]

cands = [1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49]

<script.py> output:

primes = [5, 13, 17, 29, 37, 41]

In [3]:

+100 XP

Great! Now you can do primarity checks for many more numbers! Remember, writing a function to check whether a number is prime is one of the most frequent interview questions – but it can have many variations.

**Coprime number sequence**

Two numbers aa and bb are coprime if their Greatest Common Divisor (GCD) is 1. GCD is the largest positive number that divides two given numbers aa and bb. For example, the numbers 7 and 9 are coprime because their GCD is 1.

Given two lists list1 and list2, your task is to create a new list coprimes that contains all the coprime pairs from list1 and list2.

But first, you need to write a function for the GCD using the following algorithm:

1. check if b=0b=0
   * if true, return aa as the GCD between aa and bb
   * if false, go to step 2
2. make a substitution a←ba←b and b←a%bb←a%b
3. go back to step 1

**Instructions 1/2**

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))

**Instructions 1/2**

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Define the while loop as described in the context.
* Complete the return statement.

[**Take Hint (-15 XP)**](javascript:void(0))

Create a list of tuples defining pairs of coprime numbers from list1 and list2.

def gcd(a, b):

# Define the while loop as described

while b != 0:

temp\_a = a

a = b

b = temp\_a % b

# Complete the return statement

return a

# Create a list of tuples defining pairs of coprime numbers

coprimes = [(i, j) for i in list1

for j in list2 if gcd(i, j) == 1]

print(coprimes)

<script.py> output:

[(5, 7), (5, 14), (5, 21), (5, 28), (5, 42), (5, 49), (5, 56), (5, 63), (5, 77), (5, 84), (5, 91), (5, 98), (10, 7), (10, 21), (10, 49), (10, 63), (10, 77), (10, 91), (15, 7), (15, 14), (15, 28), (15, 49), (15, 56), (15, 77), (15, 91), (15, 98), (20, 7), (20, 21), (20, 49), (20, 63), (20, 77), (20, 91), (25, 7), (25, 14), (25, 21), (25, 28), (25, 42), (25, 49), (25, 56), (25, 63), (25, 77), (25, 84), (25, 91), (25, 98), (30, 7), (30, 49), (30, 77), (30, 91), (40, 7), (40, 21), (40, 49), (40, 63), (40, 77), (40, 91), (45, 7), (45, 14), (45, 28), (45, 49), (45, 56), (45, 77), (45, 91), (45, 98), (50, 7), (50, 21), (50, 49), (50, 63), (50, 77), (50, 91), (55, 7), (55, 14), (55, 21), (55, 28), (55, 42), (55, 49), (55, 56), (55, 63), (55, 84), (55, 91), (55, 98), (60, 7), (60, 49), (60, 77), (60, 91), (65, 7), (65, 14), (65, 21), (65, 28), (65, 42), (65, 49), (65, 56), (65, 63), (65, 77), (65, 84), (65, 98)]

In [2]:

+100 XP

Good job! Writing an algorithm to find the greatest common divisor is also one of the most popular coding interview questions. Now you know how to proceed! By the way, to impress interviewers, you can substitute lines 4-6 with just one line of code a, b = b, a % b.

**Combining iterable objects**

You are given the list wlist that contains lists of different words. Your task is to create a new iterable object, where each element represents a tuple. Each tuple should contain a list from the wlist, the length of this list, and the longest word within this list. If there is ambiguity in choosing the longest word, the word with the lowest index in the considered list should be taken into account. For example, given the list

[

['dog', 'pigeon'],

['cat', 'wolf', 'seal']

]

the resulting tuples will be:  
(['dog', 'pigeon'], 2, 'pigeon')  
and  
(['cat', 'wolf', 'seal'], 3, 'wolf')

**Instructions 1/3**

**100 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))

**Instructions 1/3**

**100 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* Define a function searching for the longest word given a list of words.

[**Take Hint (-30 XP)**](javascript:void(0))

# Define a function searching for the longest word

def get\_longest\_word(words):

longest\_word = ''

for word in words:

if word in wlist:

longest\_word = word

return longest\_word

* Create a list of the lengths of each list in wlist.
* Create a list of the longest words in each list in wlist.

# Define a function searching for the longest word

def get\_longest\_word(words):

longest\_word = ''

for word in words:

if len(word) > len(longest\_word):

longest\_word = word

return longest\_word

# Create a list of the lengths of each list in wlist

lengths = [get\_longest\_word(item) for item in wlist]

# Create a list of the longest words in each list in wlist

words = [get\_longest\_word(item) for item in wlist]

Combine wlist, lengths, and words into one iterable object and print each element.

# Define a function searching for the longest word

def get\_longest\_word(words):

longest\_word = ''

for word in words:

if len(word) > len(longest\_word):

longest\_word = word

return longest\_word

# Create a list of the lengths of each list in wlist

lengths = [len(item) for item in wlist]

# Create a list of the longest words in each list in wlist

words = [get\_longest\_word(item) for item in wlist]

# Combine the resulting data into one iterable object

for item in zip(wlist, lengths, words):

print(item)

+100 XP

Good job! Don't hesitate to play around with the zip object to fully understand its properties.

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

<script.py> output:

(['Python', 'creativity', 'universe'], 3, 'creativity')

(['interview', 'study', 'job', 'university', 'lecture'], 5, 'university')

(['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research'], 7, 'programming')

<script.py> output:

(['Python', 'creativity', 'universe'], 3, 'creativity')

(['interview', 'study', 'job', 'university', 'lecture'], 5, 'university')

(['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research'], 7, 'programming')

In [1]:

**Extracting tuples**

In the previous exercise, you used two list comprehensions to create lists lengths and words that, respectively, refer to the lengths of the constituent lists in wlist and the longest words stored in those lists. In this exercise, you'll create them in a slightly different way. First, you'll need to put the same calculations into one list comprehension, which should result in a list of tuples. Second, apply the unzip operation to generate two distinct tuples, resembling lengths and words from the previous exercise.

The list wlist and the function get\_longest\_word() are already available in your workspace.

**Instructions**

**100 XP**

* Create a list of tuples each containing the length and the longest word of each item in wlist.
* Unwrap the created list and create two distinct tuples.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Did you call zip()? Yes! Wait, Not Yet! No, I have done it!

Your code generated an error. Fix it and try again!

# Create a list of tuples with lengths and longest words

result = [

(len(item), get\_longest\_word(item)) for item in wlist

]

# Unzip the result

lengths, words = list(zip(\*result))

for item in zip(wlist, lengths, words):

print(item)

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

Traceback (most recent call last):

File "script.py", line 7, in <module>

lengths, words = \_\_\_\_

NameError: name '\_\_\_\_' is not defined

In [1]: unzip

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

unzip

NameError: name 'unzip' is not defined

Traceback (most recent call last):

File "script.py", line 7, in <module>

lengths, words = zip(result)

ValueError: too many values to unpack (expected 2)

Traceback (most recent call last):

File "script.py", line 7, in <module>

lengths, words = result

ValueError: too many values to unpack (expected 2)

<script.py> output:

(['Python', 'creativity', 'universe'], 3, 'creativity')

(['interview', 'study', 'job', 'university', 'lecture'], 5, 'university')

(['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research'], 7, 'programming')

In [2]:

+100 XP

Very well! You're as master at zipping and unzipping!

**Creating a DataFrame**

Your last task in this lesson is to create a DataFrame from a dictionary supplied by a zip object. You have to take each single word stored in the list wlist and calculate its length. This data should be stored in two separate tuples that are supplied to the zip() initializer. The resulting zip object should be used to construct a DataFrame where the first column will store words and the second column will store their lengths.

The module pandas is already imported for you as pd.

**Instructions 1/4**

**25 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* [4](javascript:void(0))
* Create a list with tuples of the form (word itself, length of the word).

[**Take Hint (-7 XP)**](javascript:void(0))

**Incorrect Submission**

Your code can not be executed due to a syntax error:  
invalid syntax (script.py, line 4).

Check the first list comprehension. Did you correctly specify the iterable part? Running it generated an error: name 'wlistc' is not defined.

**Hint**

* Recall which for...in part in the list comprehension will represent the inner for loop.

# Create a list of tuples with words and their lengths

word\_lengths = [

(word, len(word)) for word in wlistc for wlistc in wlist

] #65123652165305256256256252152

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

In [1]:

File "script.py", line 4

]65123652165305256256256252152

^

SyntaxError: invalid syntax

Traceback (most recent call last):

File "script.py", line 3, in <module>

(word, len(word)) for word in wlistc for wlistc in wlist

NameError: name 'wlistc' is not defined

Traceback (most recent call last):

File "script.py", line 3, in <module>

(word, len(word)) for word in wlistc for wlistc in wlist

NameError: name 'wlistc' is not defined

In [1]: # Create a list of tuples with words and their lengths

word\_lengths = [

(word, len(word)) for word in wlistc for wlistc in wlist

]

Traceback (most recent call last):

File "<stdin>", line 3, in <module>

(word, len(word)) for word in wlistc for wlistc in wlist

NameError: name 'wlistc' is not defined

In [2]: result

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

result

NameError: name 'result' is not defined

In [3]:

* Unwrap the word\_lengths and create two separate tuples words and lengths.

[**Show Answer (-17 XP)**](javascript:void(0))

**Hint**

* Remember to use the \* prefix.

Create a zip object combining column names for the future DataFrame and the associated data.

Take Hint (-7 XP)

**Incorrect Submission**

Check your second call of zip(). Did you correctly specify the first argument? Expected something different.

**Hint**

* words and lengths should be put into a new list.
* Convert result to a dictionary and build a DataFrame.

[**Take Hint (-7 XP)**](javascript:void(0))

# Create a list of tuples with words and their lengths

word\_lengths = [

(item, len(item)) for items in wlist for item in items

]

# Unwrap the word\_lengths

words, lengths = zip(\*word\_lengths)

# Create a zip object

col\_names = ['word', 'length']

result = zip(col\_names, [words, lengths])

# Convert the result to a dictionary and build a DataFrame

data\_frame = \_\_\_\_

print(data\_frame)

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

In [3]:

* Convert result to a dictionary and build a DataFrame.

[**Take Hint (-7 XP)**](javascript:void(0))

# Create a list of tuples with words and their lengths

word\_lengths = [

(item, len(item)) for items in wlist for item in items

]

# Unwrap the word\_lengths

words, lengths = zip(\*word\_lengths)

# Create a zip object

col\_names = ['word', 'length']

result = zip(col\_names, [words, lengths])

# Convert the result to a dictionary and build a DataFrame

data\_frame = pd.DataFrame(dict(result))

print(data\_frame)

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

wlist = [['Python', 'creativity', 'universe'], ['interview', 'study', 'job', 'university', 'lecture'], ['task', 'objective', 'aim', 'subject', 'programming', 'test', 'research']]

<script.py> output:

word length

0 Python 6

1 creativity 10

2 universe 8

3 interview 9

4 study 5

5 job 3

6 university 10

7 lecture 7

8 task 4

9 objective 9

10 aim 3

11 subject 7

12 programming 11

13 test 4

14 research 8

In [3]:

+45 XP

Great! You practiced the workflow to create a DataFrame from a zip object.

**Exercise**

**Exercise**

**Shift a string**

You're going to create a generator that, given a string, produces a sequence of constituent characters shifted by a specified number of positions shift. For example, the string 'sushi' produces the sequence 'h', 'i', 's', 'u', 's' when we shift by 2 positions to the right (shift = 2). When we shift by 2 positions to the left (shift = -2), the resulting sequence will be 's', 'h', 'i', 's', 'u'.

Tip: use the % operator to cycle through the valid indices. Applying it to a positive or negative number gives a non-negative remainder, which can be helpful when shifting your index. For example:

* -2 % 10 = 8
* -1 % 10 = 9
* 0 % 10 = 0
* 1 % 10 = 1
* 2 % 10 = 2

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Loop over the indices of a string in an increasing order.
* Find which character will correspond to the index when we perform the shift.
* Create a generator that shifts the string 'DataCamp' by 3 positions to the right (i.e. "ampDataC").
* Create a new string using the generator and print it out.

[**Take Hint (-30 XP)**](javascript:void(0))

[**Show Answer (-70 XP)**](javascript:void(0))

**Hint**

* Use the range() function to loop over the indices.
* To retrieve a character, you must calculate a remainder that will serve as an index.
* You can use .join() to create a string from a list of strings.

Awesome, thanks for your feedback!

def shift\_string(string, shift):

len\_string = len(string)

# Loop over the indices of a string

for idx in range(len\_string):

# Find which character will correspond to the index.

string[idx] = string[idx % len\_string]

# Create a generator

gen = shift\_string('ampDataC', 3)

# Create a new string using the generator and print it out

string\_shifted = ''.join(gen)

print(string\_shifted)



**solution.py**

def shift\_string(string, shift):

len\_string = len(string)

# Loop over the indices of a string

for idx in range(0, len\_string):

# Find which character will correspond to the index.

yield string[(idx - shift) % len\_string]

# Create a generator

gen = shift\_string('DataCamp', 3)

# Create a new string using the generator and print it out

string\_shifted = ''.join(gen)

print(string\_shifted)

In [1]: def shift\_string(string, shift):

len\_string = len(string)

# Loop over the indices of a string

for idx in range(len\_string):

# Find which character will correspond to the index.

string[idx] = string[idx % len\_string]

# Create a generator

gen = shift\_string('ampDataC', 3)

# Create a new string using the generator and print it out

string\_shifted = ''.join(gen)

print(string\_shifted)

Traceback (most recent call last):

File "<stdin>", line 9, in <module>

gen = shift\_string('ampDataC', 3)

File "<stdin>", line 6, in shift\_string

string[idx] = string[idx % len\_string]

TypeError: 'str' object does not support item assignment

<script.py> output:

ampDataC

In [2]:

+0 XP

Very well! As you might have noticed, this generator function can be used to shift any indexable object, not only strings.

**Exercise**

**Exercise**

**Throw a dice**

Let's create an infinite generator! Your task is to define the simulate\_dice\_throws() generator. It generates the outcomes of a 6-sided dice tosses in the form of a dictionary out. Each key is a possible outcome (1, 2, 3, 4, 5, 6). Each value is a list: the first value is the amount of realizations of an outcome and the second, the ratio of realizations to the total number of tosses total.

Tip: use the randint() function from the random module (already imported). It generates a random integer in the specified interval (*e.g.* randint(1, 2) can be 1 or 2).

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Simulate a single toss to get a new number.
* Update the number and the ratio of realization.
* Yield the updated dictionary.
* Create the generator and simulate 1000 tosses.

[**Take Hint (-30 XP)**](javascript:void(0))

def simulate\_dice\_throws():

total, out = 0, dict([(i, [0, 0]) for i in range(1, 7)])

while True:

# Simulate a single toss to get a new number

num = random.randint(1, 6)

total += 1

# Update the number and the ratio of realizations

out[num][0] = out[num][0] + 1

for j in range(1, 7):

out[j][1] = round(out[j][0]/total, 2)

# Yield the updated dictionary

yield out

# Create the generator and simulate 1000 tosses

dice\_simulator = simulate\_dice\_throws()

for i in range(1, 1001):

print(str(i) + ': ' + str(next(dice\_simulator)))

<script.py> output:

1: {1: [0, 0.0], 2: [0, 0.0], 3: [0, 0.0], 4: [0, 0.0], 5: [0, 0.0], 6: [1, 1.0]}

2: {1: [1, 0.5], 2: [0, 0.0], 3: [0, 0.0], 4: [0, 0.0], 5: [0, 0.0], 6: [1, 0.5]}

3: {1: [2, 0.67], 2: [0, 0.0], 3: [0, 0.0], 4: [0, 0.0], 5: [0, 0.0], 6: [1, 0.33]}

4: {1: [2, 0.5], 2: [0, 0.0], 3: [0, 0.0], 4: [0, 0.0], 5: [0, 0.0], 6: [2, 0.5]}

5: {1: [2, 0.4], 2: [0, 0.0], 3: [1, 0.2], 4: [0, 0.0], 5: [0, 0.0], 6: [2, 0.4]}

6: {1: [2, 0.33], 2: [1, 0.17], 3: [1, 0.17], 4: [0, 0.0], 5: [0, 0.0], 6: [2, 0.33]}

7: {1: [2, 0.29], 2: [2, 0.29], 3: [1, 0.14], 4: [0, 0.0], 5: [0, 0.0], 6: [2, 0.29]}

8: {1: [2, 0.25], 2: [3, 0.38], 3: [1, 0.12], 4: [0, 0.0], 5: [0, 0.0], 6: [2, 0.25]}

9: {1: [2, 0.22], 2: [3, 0.33], 3: [1, 0.11], 4: [0, 0.0], 5: [0, 0.0], 6: [3, 0.33]}

10: {1: [3, 0.3], 2: [3, 0.3], 3: [1, 0.1], 4: [0, 0.0], 5: [0, 0.0], 6: [3, 0.3]}

11: {1: [3, 0.27], 2: [3, 0.27], 3: [1, 0.09], 4: [0, 0.0], 5: [0, 0.0], 6: [4, 0.36]}

12: {1: [3, 0.25], 2: [3, 0.25], 3: [1, 0.08], 4: [0, 0.0], 5: [0, 0.0], 6: [5, 0.42]}

13: {1: [3, 0.23], 2: [3, 0.23], 3: [1, 0.08], 4: [0, 0.0], 5: [1, 0.08], 6: [5, 0.38]}

14: {1: [4, 0.29], 2: [3, 0.21], 3: [1, 0.07], 4: [0, 0.0], 5: [1, 0.07], 6: [5, 0.36]}

15: {1: [4, 0.27], 2: [3, 0.2], 3: [1, 0.07], 4: [0, 0.0], 5: [2, 0.13], 6: [5, 0.33]}

16: {1: [4, 0.25], 2: [3, 0.19], 3: [1, 0.06], 4: [1, 0.06], 5: [2, 0.12], 6: [5, 0.31]}

17: {1: [5, 0.29], 2: [3, 0.18], 3: [1, 0.06], 4: [1, 0.06], 5: [2, 0.12], 6: [5, 0.29]}

18: {1: [6, 0.33], 2: [3, 0.17], 3: [1, 0.06], 4: [1, 0.06], 5: [2, 0.11], 6: [5, 0.28]}

19: {1: [7, 0.37], 2: [3, 0.16], 3: [1, 0.05], 4: [1, 0.05], 5: [2, 0.11], 6: [5, 0.26]}

20: {1: [7, 0.35], 2: [4, 0.2], 3: [1, 0.05], 4: [1, 0.05], 5: [2, 0.1], 6: [5, 0.25]}

21: {1: [7, 0.33], 2: [5, 0.24], 3: [1, 0.05], 4: [1, 0.05], 5: [2, 0.1], 6: [5, 0.24]}

22: {1: [7, 0.32], 2: [5, 0.23], 3: [1, 0.05], 4: [1, 0.05], 5: [3, 0.14], 6: [5, 0.23]}

23: {1: [7, 0.3], 2: [5, 0.22], 3: [1, 0.04], 4: [1, 0.04], 5: [4, 0.17], 6: [5, 0.22]}

24: {1: [8, 0.33], 2: [5, 0.21], 3: [1, 0.04], 4: [1, 0.04], 5: [4, 0.17], 6: [5, 0.21]}

25: {1: [8, 0.32], 2: [5, 0.2], 3: [1, 0.04], 4: [1, 0.04], 5: [5, 0.2], 6: [5, 0.2]}

26: {1: [8, 0.31], 2: [6, 0.23], 3: [1, 0.04], 4: [1, 0.04], 5: [5, 0.19], 6: [5, 0.19]}

27: {1: [8, 0.3], 2: [6, 0.22], 3: [1, 0.04], 4: [1, 0.04], 5: [5, 0.19], 6: [6, 0.22]}

28: {1: [8, 0.29], 2: [6, 0.21], 3: [1, 0.04], 4: [1, 0.04], 5: [5, 0.18], 6: [7, 0.25]}

29: {1: [8, 0.28], 2: [6, 0.21], 3: [1, 0.03], 4: [1, 0.03], 5: [5, 0.17], 6: [8, 0.28]}

30: {1: [8, 0.27], 2: [6, 0.2], 3: [1, 0.03], 4: [1, 0.03], 5: [6, 0.2], 6: [8, 0.27]}

31: {1: [8, 0.26], 2: [6, 0.19], 3: [1, 0.03], 4: [2, 0.06], 5: [6, 0.19], 6: [8, 0.26]}

32: {1: [8, 0.25], 2: [7, 0.22], 3: [1, 0.03], 4: [2, 0.06], 5: [6, 0.19], 6: [8, 0.25]}

33: {1: [8, 0.24], 2: [7, 0.21], 3: [1, 0.03], 4: [3, 0.09], 5: [6, 0.18], 6: [8, 0.24]}

34: {1: [8, 0.24], 2: [7, 0.21], 3: [1, 0.03], 4: [3, 0.09], 5: [7, 0.21], 6: [8, 0.24]}

35: {1: [8, 0.23], 2: [7, 0.2], 3: [2, 0.06], 4: [3, 0.09], 5: [7, 0.2], 6: [8, 0.23]}

36: {1: [9, 0.25], 2: [7, 0.19], 3: [2, 0.06], 4: [3, 0.08], 5: [7, 0.19], 6: [8, 0.22]}

37: {1: [9, 0.24], 2: [8, 0.22], 3: [2, 0.05], 4: [3, 0.08], 5: [7, 0.19], 6: [8, 0.22]}

38: {1: [9, 0.24], 2: [8, 0.21], 3: [2, 0.05], 4: [3, 0.08], 5: [7, 0.18], 6: [9, 0.24]}

39: {1: [9, 0.23], 2: [8, 0.21], 3: [2, 0.05], 4: [4, 0.1], 5: [7, 0.18], 6: [9, 0.23]}

40: {1: [9, 0.23], 2: [8, 0.2], 3: [3, 0.07], 4: [4, 0.1], 5: [7, 0.17], 6: [9, 0.23]}

41: {1: [9, 0.22], 2: [8, 0.2], 3: [4, 0.1], 4: [4, 0.1], 5: [7, 0.17], 6: [9, 0.22]}

42: {1: [9, 0.21], 2: [9, 0.21], 3: [4, 0.1], 4: [4, 0.1], 5: [7, 0.17], 6: [9, 0.21]}

43: {1: [9, 0.21], 2: [10, 0.23], 3: [4, 0.09], 4: [4, 0.09], 5: [7, 0.16], 6: [9, 0.21]}

44: {1: [9, 0.2], 2: [10, 0.23], 3: [5, 0.11], 4: [4, 0.09], 5: [7, 0.16], 6: [9, 0.2]}

45: {1: [10, 0.22], 2: [10, 0.22], 3: [5, 0.11], 4: [4, 0.09], 5: [7, 0.16], 6: [9, 0.2]}

46: {1: [11, 0.24], 2: [10, 0.22], 3: [5, 0.11], 4: [4, 0.09], 5: [7, 0.15], 6: [9, 0.2]}

47: {1: [11, 0.23], 2: [10, 0.21], 3: [5, 0.11], 4: [5, 0.11], 5: [7, 0.15], 6: [9, 0.19]}

48: {1: [12, 0.25], 2: [10, 0.21], 3: [5, 0.1], 4: [5, 0.1], 5: [7, 0.15], 6: [9, 0.19]}

49: {1: [12, 0.24], 2: [10, 0.2], 3: [6, 0.12], 4: [5, 0.1], 5: [7, 0.14], 6: [9, 0.18]}

50: {1: [12, 0.24], 2: [10, 0.2], 3: [7, 0.14], 4: [5, 0.1], 5: [7, 0.14], 6: [9, 0.18]}

51: {1: [12, 0.24], 2: [10, 0.2], 3: [7, 0.14], 4: [5, 0.1], 5: [8, 0.16], 6: [9, 0.18]}

52: {1: [12, 0.23], 2: [10, 0.19], 3: [8, 0.15], 4: [5, 0.1], 5: [8, 0.15], 6: [9, 0.17]}

53: {1: [13, 0.25], 2: [10, 0.19], 3: [8, 0.15], 4: [5, 0.09], 5: [8, 0.15], 6: [9, 0.17]}

54: {1: [13, 0.24], 2: [10, 0.19], 3: [8, 0.15], 4: [5, 0.09], 5: [8, 0.15], 6: [10, 0.19]}

55: {1: [13, 0.24], 2: [10, 0.18], 3: [8, 0.15], 4: [6, 0.11], 5: [8, 0.15], 6: [10, 0.18]}

56: {1: [13, 0.23], 2: [10, 0.18], 3: [8, 0.14], 4: [6, 0.11], 5: [9, 0.16], 6: [10, 0.18]}

57: {1: [14, 0.25], 2: [10, 0.18], 3: [8, 0.14], 4: [6, 0.11], 5: [9, 0.16], 6: [10, 0.18]}

58: {1: [14, 0.24], 2: [10, 0.17], 3: [8, 0.14], 4: [7, 0.12], 5: [9, 0.16], 6: [10, 0.17]}

59: {1: [15, 0.25], 2: [10, 0.17], 3: [8, 0.14], 4: [7, 0.12], 5: [9, 0.15], 6: [10, 0.17]}

60: {1: [15, 0.25], 2: [10, 0.17], 3: [8, 0.13], 4: [7, 0.12], 5: [10, 0.17], 6: [10, 0.17]}

61: {1: [15, 0.25], 2: [10, 0.16], 3: [9, 0.15], 4: [7, 0.11], 5: [10, 0.16], 6: [10, 0.16]}

62: {1: [15, 0.24], 2: [10, 0.16], 3: [9, 0.15], 4: [7, 0.11], 5: [10, 0.16], 6: [11, 0.18]}

63: {1: [15, 0.24], 2: [10, 0.16], 3: [9, 0.14], 4: [7, 0.11], 5: [11, 0.17], 6: [11, 0.17]}

64: {1: [15, 0.23], 2: [10, 0.16], 3: [10, 0.16], 4: [7, 0.11], 5: [11, 0.17], 6: [11, 0.17]}

65: {1: [15, 0.23], 2: [10, 0.15], 3: [10, 0.15], 4: [7, 0.11], 5: [12, 0.18], 6: [11, 0.17]}

66: {1: [15, 0.23], 2: [11, 0.17], 3: [10, 0.15], 4: [7, 0.11], 5: [12, 0.18], 6: [11, 0.17]}

67: {1: [15, 0.22], 2: [11, 0.16], 3: [10, 0.15], 4: [7, 0.1], 5: [12, 0.18], 6: [12, 0.18]}

68: {1: [16, 0.24], 2: [11, 0.16], 3: [10, 0.15], 4: [7, 0.1], 5: [12, 0.18], 6: [12, 0.18]}

69: {1: [17, 0.25], 2: [11, 0.16], 3: [10, 0.14], 4: [7, 0.1], 5: [12, 0.17], 6: [12, 0.17]}

70: {1: [17, 0.24], 2: [11, 0.16], 3: [10, 0.14], 4: [7, 0.1], 5: [12, 0.17], 6: [13, 0.19]}

71: {1: [17, 0.24], 2: [12, 0.17], 3: [10, 0.14], 4: [7, 0.1], 5: [12, 0.17], 6: [13, 0.18]}

72: {1: [17, 0.24], 2: [12, 0.17], 3: [11, 0.15], 4: [7, 0.1], 5: [12, 0.17], 6: [13, 0.18]}

73: {1: [18, 0.25], 2: [12, 0.16], 3: [11, 0.15], 4: [7, 0.1], 5: [12, 0.16], 6: [13, 0.18]}

74: {1: [18, 0.24], 2: [13, 0.18], 3: [11, 0.15], 4: [7, 0.09], 5: [12, 0.16], 6: [13, 0.18]}

75: {1: [19, 0.25], 2: [13, 0.17], 3: [11, 0.15], 4: [7, 0.09], 5: [12, 0.16], 6: [13, 0.17]}

76: {1: [19, 0.25], 2: [13, 0.17], 3: [11, 0.14], 4: [8, 0.11], 5: [12, 0.16], 6: [13, 0.17]}

77: {1: [19, 0.25], 2: [13, 0.17], 3: [12, 0.16], 4: [8, 0.1], 5: [12, 0.16], 6: [13, 0.17]}

78: {1: [19, 0.24], 2: [13, 0.17], 3: [12, 0.15], 4: [9, 0.12], 5: [12, 0.15], 6: [13, 0.17]}

79: {1: [19, 0.24], 2: [13, 0.16], 3: [12, 0.15], 4: [9, 0.11], 5: [12, 0.15], 6: [14, 0.18]}

80: {1: [19, 0.24], 2: [13, 0.16], 3: [13, 0.16], 4: [9, 0.11], 5: [12, 0.15], 6: [14, 0.17]}

81: {1: [19, 0.23], 2: [14, 0.17], 3: [13, 0.16], 4: [9, 0.11], 5: [12, 0.15], 6: [14, 0.17]}

82: {1: [19, 0.23], 2: [14, 0.17], 3: [14, 0.17], 4: [9, 0.11], 5: [12, 0.15], 6: [14, 0.17]}

83: {1: [19, 0.23], 2: [14, 0.17], 3: [15, 0.18], 4: [9, 0.11], 5: [12, 0.14], 6: [14, 0.17]}

84: {1: [19, 0.23], 2: [15, 0.18], 3: [15, 0.18], 4: [9, 0.11], 5: [12, 0.14], 6: [14, 0.17]}

85: {1: [19, 0.22], 2: [15, 0.18], 3: [15, 0.18], 4: [9, 0.11], 5: [12, 0.14], 6: [15, 0.18]}

86: {1: [19, 0.22], 2: [15, 0.17], 3: [16, 0.19], 4: [9, 0.1], 5: [12, 0.14], 6: [15, 0.17]}

87: {1: [19, 0.22], 2: [15, 0.17], 3: [16, 0.18], 4: [9, 0.1], 5: [12, 0.14], 6: [16, 0.18]}

88: {1: [19, 0.22], 2: [15, 0.17], 3: [16, 0.18], 4: [9, 0.1], 5: [12, 0.14], 6: [17, 0.19]}

89: {1: [19, 0.21], 2: [15, 0.17], 3: [16, 0.18], 4: [9, 0.1], 5: [12, 0.13], 6: [18, 0.2]}

90: {1: [20, 0.22], 2: [15, 0.17], 3: [16, 0.18], 4: [9, 0.1], 5: [12, 0.13], 6: [18, 0.2]}

91: {1: [20, 0.22], 2: [15, 0.16], 3: [16, 0.18], 4: [9, 0.1], 5: [13, 0.14], 6: [18, 0.2]}

92: {1: [20, 0.22], 2: [15, 0.16], 3: [16, 0.17], 4: [9, 0.1], 5: [13, 0.14], 6: [19, 0.21]}

93: {1: [20, 0.22], 2: [16, 0.17], 3: [16, 0.17], 4: [9, 0.1], 5: [13, 0.14], 6: [19, 0.2]}

94: {1: [20, 0.21], 2: [16, 0.17], 3: [16, 0.17], 4: [9, 0.1], 5: [14, 0.15], 6: [19, 0.2]}

95: {1: [20, 0.21], 2: [16, 0.17], 3: [16, 0.17], 4: [9, 0.09], 5: [14, 0.15], 6: [20, 0.21]}

96: {1: [20, 0.21], 2: [17, 0.18], 3: [16, 0.17], 4: [9, 0.09], 5: [14, 0.15], 6: [20, 0.21]}

97: {1: [20, 0.21], 2: [18, 0.19], 3: [16, 0.16], 4: [9, 0.09], 5: [14, 0.14], 6: [20, 0.21]}

98: {1: [20, 0.2], 2: [18, 0.18], 3: [16, 0.16], 4: [10, 0.1], 5: [14, 0.14], 6: [20, 0.2]}

99: {1: [20, 0.2], 2: [18, 0.18], 3: [16, 0.16], 4: [11, 0.11], 5: [14, 0.14], 6: [20, 0.2]}

100: {1: [20, 0.2], 2: [18, 0.18], 3: [17, 0.17], 4: [11, 0.11], 5: [14, 0.14], 6: [20, 0.2]}

101: {1: [20, 0.2], 2: [18, 0.18], 3: [17, 0.17], 4: [11, 0.11], 5: [14, 0.14], 6: [21, 0.21]}

102: {1: [20, 0.2], 2: [18, 0.18], 3: [17, 0.17], 4: [11, 0.11], 5: [14, 0.14], 6: [22, 0.22]}

103: {1: [20, 0.19], 2: [18, 0.17], 3: [17, 0.17], 4: [11, 0.11], 5: [15, 0.15], 6: [22, 0.21]}

104: {1: [20, 0.19], 2: [19, 0.18], 3: [17, 0.16], 4: [11, 0.11], 5: [15, 0.14], 6: [22, 0.21]}

105: {1: [20, 0.19], 2: [19, 0.18], 3: [17, 0.16], 4: [11, 0.1], 5: [15, 0.14], 6: [23, 0.22]}

106: {1: [20, 0.19], 2: [19, 0.18], 3: [18, 0.17], 4: [11, 0.1], 5: [15, 0.14], 6: [23, 0.22]}

107: {1: [21, 0.2], 2: [19, 0.18], 3: [18, 0.17], 4: [11, 0.1], 5: [15, 0.14], 6: [23, 0.21]}

108: {1: [21, 0.19], 2: [20, 0.19], 3: [18, 0.17], 4: [11, 0.1], 5: [15, 0.14], 6: [23, 0.21]}

109: {1: [22, 0.2], 2: [20, 0.18], 3: [18, 0.17], 4: [11, 0.1], 5: [15, 0.14], 6: [23, 0.21]}

110: {1: [22, 0.2], 2: [20, 0.18], 3: [19, 0.17], 4: [11, 0.1], 5: [15, 0.14], 6: [23, 0.21]}

111: {1: [22, 0.2], 2: [20, 0.18], 3: [19, 0.17], 4: [12, 0.11], 5: [15, 0.14], 6: [23, 0.21]}

112: {1: [22, 0.2], 2: [20, 0.18], 3: [20, 0.18], 4: [12, 0.11], 5: [15, 0.13], 6: [23, 0.21]}

113: {1: [23, 0.2], 2: [20, 0.18], 3: [20, 0.18], 4: [12, 0.11], 5: [15, 0.13], 6: [23, 0.2]}

114: {1: [23, 0.2], 2: [21, 0.18], 3: [20, 0.18], 4: [12, 0.11], 5: [15, 0.13], 6: [23, 0.2]}

115: {1: [23, 0.2], 2: [21, 0.18], 3: [20, 0.17], 4: [12, 0.1], 5: [16, 0.14], 6: [23, 0.2]}

116: {1: [23, 0.2], 2: [21, 0.18], 3: [20, 0.17], 4: [12, 0.1], 5: [16, 0.14], 6: [24, 0.21]}

117: {1: [23, 0.2], 2: [21, 0.18], 3: [21, 0.18], 4: [12, 0.1], 5: [16, 0.14], 6: [24, 0.21]}

118: {1: [23, 0.19], 2: [22, 0.19], 3: [21, 0.18], 4: [12, 0.1], 5: [16, 0.14], 6: [24, 0.2]}

119: {1: [23, 0.19], 2: [22, 0.18], 3: [21, 0.18], 4: [12, 0.1], 5: [16, 0.13], 6: [25, 0.21]}

120: {1: [23, 0.19], 2: [22, 0.18], 3: [21, 0.17], 4: [13, 0.11], 5: [16, 0.13], 6: [25, 0.21]}

121: {1: [23, 0.19], 2: [22, 0.18], 3: [21, 0.17], 4: [14, 0.12], 5: [16, 0.13], 6: [25, 0.21]}

122: {1: [23, 0.19], 2: [22, 0.18], 3: [21, 0.17], 4: [14, 0.11], 5: [16, 0.13], 6: [26, 0.21]}

123: {1: [23, 0.19], 2: [22, 0.18], 3: [21, 0.17], 4: [15, 0.12], 5: [16, 0.13], 6: [26, 0.21]}

124: {1: [23, 0.19], 2: [23, 0.19], 3: [21, 0.17], 4: [15, 0.12], 5: [16, 0.13], 6: [26, 0.21]}

125: {1: [23, 0.18], 2: [23, 0.18], 3: [22, 0.18], 4: [15, 0.12], 5: [16, 0.13], 6: [26, 0.21]}

126: {1: [23, 0.18], 2: [24, 0.19], 3: [22, 0.17], 4: [15, 0.12], 5: [16, 0.13], 6: [26, 0.21]}

127: {1: [23, 0.18], 2: [25, 0.2], 3: [22, 0.17], 4: [15, 0.12], 5: [16, 0.13], 6: [26, 0.2]}

128: {1: [23, 0.18], 2: [25, 0.2], 3: [22, 0.17], 4: [15, 0.12], 5: [16, 0.12], 6: [27, 0.21]}

129: {1: [23, 0.18], 2: [25, 0.19], 3: [22, 0.17], 4: [15, 0.12], 5: [17, 0.13], 6: [27, 0.21]}

130: {1: [23, 0.18], 2: [25, 0.19], 3: [22, 0.17], 4: [15, 0.12], 5: [18, 0.14], 6: [27, 0.21]}

131: {1: [23, 0.18], 2: [25, 0.19], 3: [23, 0.18], 4: [15, 0.11], 5: [18, 0.14], 6: [27, 0.21]}

132: {1: [23, 0.17], 2: [25, 0.19], 3: [23, 0.17], 4: [15, 0.11], 5: [18, 0.14], 6: [28, 0.21]}

133: {1: [23, 0.17], 2: [25, 0.19], 3: [23, 0.17], 4: [15, 0.11], 5: [19, 0.14], 6: [28, 0.21]}

134: {1: [23, 0.17], 2: [25, 0.19], 3: [23, 0.17], 4: [16, 0.12], 5: [19, 0.14], 6: [28, 0.21]}

135: {1: [23, 0.17], 2: [25, 0.19], 3: [23, 0.17], 4: [16, 0.12], 5: [20, 0.15], 6: [28, 0.21]}

136: {1: [23, 0.17], 2: [25, 0.18], 3: [23, 0.17], 4: [17, 0.12], 5: [20, 0.15], 6: [28, 0.21]}

137: {1: [23, 0.17], 2: [25, 0.18], 3: [24, 0.18], 4: [17, 0.12], 5: [20, 0.15], 6: [28, 0.2]}

138: {1: [23, 0.17], 2: [26, 0.19], 3: [24, 0.17], 4: [17, 0.12], 5: [20, 0.14], 6: [28, 0.2]}

139: {1: [23, 0.17], 2: [27, 0.19], 3: [24, 0.17], 4: [17, 0.12], 5: [20, 0.14], 6: [28, 0.2]}

140: {1: [23, 0.16], 2: [27, 0.19], 3: [24, 0.17], 4: [17, 0.12], 5: [21, 0.15], 6: [28, 0.2]}

141: {1: [23, 0.16], 2: [27, 0.19], 3: [24, 0.17], 4: [18, 0.13], 5: [21, 0.15], 6: [28, 0.2]}

142: {1: [24, 0.17], 2: [27, 0.19], 3: [24, 0.17], 4: [18, 0.13], 5: [21, 0.15], 6: [28, 0.2]}

143: {1: [25, 0.17], 2: [27, 0.19], 3: [24, 0.17], 4: [18, 0.13], 5: [21, 0.15], 6: [28, 0.2]}

144: {1: [26, 0.18], 2: [27, 0.19], 3: [24, 0.17], 4: [18, 0.12], 5: [21, 0.15], 6: [28, 0.19]}

145: {1: [26, 0.18], 2: [28, 0.19], 3: [24, 0.17], 4: [18, 0.12], 5: [21, 0.14], 6: [28, 0.19]}

146: {1: [26, 0.18], 2: [28, 0.19], 3: [24, 0.16], 4: [18, 0.12], 5: [21, 0.14], 6: [29, 0.2]}

147: {1: [26, 0.18], 2: [29, 0.2], 3: [24, 0.16], 4: [18, 0.12], 5: [21, 0.14], 6: [29, 0.2]}

148: {1: [26, 0.18], 2: [29, 0.2], 3: [24, 0.16], 4: [18, 0.12], 5: [21, 0.14], 6: [30, 0.2]}

149: {1: [26, 0.17], 2: [29, 0.19], 3: [24, 0.16], 4: [19, 0.13], 5: [21, 0.14], 6: [30, 0.2]}

150: {1: [26, 0.17], 2: [29, 0.19], 3: [24, 0.16], 4: [19, 0.13], 5: [22, 0.15], 6: [30, 0.2]}

151: {1: [27, 0.18], 2: [29, 0.19], 3: [24, 0.16], 4: [19, 0.13], 5: [22, 0.15], 6: [30, 0.2]}

152: {1: [27, 0.18], 2: [29, 0.19], 3: [24, 0.16], 4: [20, 0.13], 5: [22, 0.14], 6: [30, 0.2]}

153: {1: [27, 0.18], 2: [29, 0.19], 3: [24, 0.16], 4: [21, 0.14], 5: [22, 0.14], 6: [30, 0.2]}

154: {1: [27, 0.18], 2: [29, 0.19], 3: [24, 0.16], 4: [21, 0.14], 5: [23, 0.15], 6: [30, 0.19]}

155: {1: [27, 0.17], 2: [29, 0.19], 3: [24, 0.15], 4: [22, 0.14], 5: [23, 0.15], 6: [30, 0.19]}

156: {1: [27, 0.17], 2: [29, 0.19], 3: [24, 0.15], 4: [22, 0.14], 5: [24, 0.15], 6: [30, 0.19]}

157: {1: [27, 0.17], 2: [29, 0.18], 3: [25, 0.16], 4: [22, 0.14], 5: [24, 0.15], 6: [30, 0.19]}

158: {1: [27, 0.17], 2: [29, 0.18], 3: [25, 0.16], 4: [22, 0.14], 5: [25, 0.16], 6: [30, 0.19]}

159: {1: [28, 0.18], 2: [29, 0.18], 3: [25, 0.16], 4: [22, 0.14], 5: [25, 0.16], 6: [30, 0.19]}

160: {1: [28, 0.17], 2: [29, 0.18], 3: [25, 0.16], 4: [22, 0.14], 5: [25, 0.16], 6: [31, 0.19]}

161: {1: [28, 0.17], 2: [29, 0.18], 3: [25, 0.16], 4: [22, 0.14], 5: [25, 0.16], 6: [32, 0.2]}

162: {1: [29, 0.18], 2: [29, 0.18], 3: [25, 0.15], 4: [22, 0.14], 5: [25, 0.15], 6: [32, 0.2]}

163: {1: [29, 0.18], 2: [29, 0.18], 3: [25, 0.15], 4: [22, 0.13], 5: [25, 0.15], 6: [33, 0.2]}

164: {1: [29, 0.18], 2: [29, 0.18], 3: [25, 0.15], 4: [22, 0.13], 5: [26, 0.16], 6: [33, 0.2]}

165: {1: [29, 0.18], 2: [29, 0.18], 3: [26, 0.16], 4: [22, 0.13], 5: [26, 0.16], 6: [33, 0.2]}

166: {1: [29, 0.17], 2: [29, 0.17], 3: [26, 0.16], 4: [22, 0.13], 5: [26, 0.16], 6: [34, 0.2]}

167: {1: [29, 0.17], 2: [29, 0.17], 3: [27, 0.16], 4: [22, 0.13], 5: [26, 0.16], 6: [34, 0.2]}

168: {1: [30, 0.18], 2: [29, 0.17], 3: [27, 0.16], 4: [22, 0.13], 5: [26, 0.15], 6: [34, 0.2]}

169: {1: [30, 0.18], 2: [29, 0.17], 3: [28, 0.17], 4: [22, 0.13], 5: [26, 0.15], 6: [34, 0.2]}

170: {1: [30, 0.18], 2: [29, 0.17], 3: [28, 0.16], 4: [23, 0.14], 5: [26, 0.15], 6: [34, 0.2]}

171: {1: [30, 0.18], 2: [30, 0.18], 3: [28, 0.16], 4: [23, 0.13], 5: [26, 0.15], 6: [34, 0.2]}

172: {1: [30, 0.17], 2: [30, 0.17], 3: [28, 0.16], 4: [24, 0.14], 5: [26, 0.15], 6: [34, 0.2]}

173: {1: [31, 0.18], 2: [30, 0.17], 3: [28, 0.16], 4: [24, 0.14], 5: [26, 0.15], 6: [34, 0.2]}

174: {1: [31, 0.18], 2: [30, 0.17], 3: [28, 0.16], 4: [24, 0.14], 5: [26, 0.15], 6: [35, 0.2]}

175: {1: [31, 0.18], 2: [30, 0.17], 3: [28, 0.16], 4: [24, 0.14], 5: [26, 0.15], 6: [36, 0.21]}

176: {1: [31, 0.18], 2: [30, 0.17], 3: [29, 0.16], 4: [24, 0.14], 5: [26, 0.15], 6: [36, 0.2]}

177: {1: [31, 0.18], 2: [30, 0.17], 3: [29, 0.16], 4: [24, 0.14], 5: [27, 0.15], 6: [36, 0.2]}

178: {1: [31, 0.17], 2: [31, 0.17], 3: [29, 0.16], 4: [24, 0.13], 5: [27, 0.15], 6: [36, 0.2]}

179: {1: [31, 0.17], 2: [31, 0.17], 3: [29, 0.16], 4: [24, 0.13], 5: [28, 0.16], 6: [36, 0.2]}

180: {1: [32, 0.18], 2: [31, 0.17], 3: [29, 0.16], 4: [24, 0.13], 5: [28, 0.16], 6: [36, 0.2]}

181: {1: [32, 0.18], 2: [31, 0.17], 3: [29, 0.16], 4: [24, 0.13], 5: [28, 0.15], 6: [37, 0.2]}

182: {1: [32, 0.18], 2: [31, 0.17], 3: [30, 0.16], 4: [24, 0.13], 5: [28, 0.15], 6: [37, 0.2]}

183: {1: [32, 0.17], 2: [31, 0.17], 3: [30, 0.16], 4: [24, 0.13], 5: [28, 0.15], 6: [38, 0.21]}

184: {1: [32, 0.17], 2: [31, 0.17], 3: [30, 0.16], 4: [24, 0.13], 5: [29, 0.16], 6: [38, 0.21]}

185: {1: [32, 0.17], 2: [31, 0.17], 3: [30, 0.16], 4: [24, 0.13], 5: [30, 0.16], 6: [38, 0.21]}

186: {1: [32, 0.17], 2: [32, 0.17], 3: [30, 0.16], 4: [24, 0.13], 5: [30, 0.16], 6: [38, 0.2]}

187: {1: [32, 0.17], 2: [33, 0.18], 3: [30, 0.16], 4: [24, 0.13], 5: [30, 0.16], 6: [38, 0.2]}

188: {1: [32, 0.17], 2: [33, 0.18], 3: [31, 0.16], 4: [24, 0.13], 5: [30, 0.16], 6: [38, 0.2]}

189: {1: [32, 0.17], 2: [34, 0.18], 3: [31, 0.16], 4: [24, 0.13], 5: [30, 0.16], 6: [38, 0.2]}

190: {1: [32, 0.17], 2: [34, 0.18], 3: [31, 0.16], 4: [24, 0.13], 5: [31, 0.16], 6: [38, 0.2]}

191: {1: [32, 0.17], 2: [34, 0.18], 3: [31, 0.16], 4: [24, 0.13], 5: [32, 0.17], 6: [38, 0.2]}

192: {1: [33, 0.17], 2: [34, 0.18], 3: [31, 0.16], 4: [24, 0.12], 5: [32, 0.17], 6: [38, 0.2]}

193: {1: [33, 0.17], 2: [34, 0.18], 3: [31, 0.16], 4: [24, 0.12], 5: [33, 0.17], 6: [38, 0.2]}

194: {1: [33, 0.17], 2: [34, 0.18], 3: [32, 0.16], 4: [24, 0.12], 5: [33, 0.17], 6: [38, 0.2]}

195: {1: [33, 0.17], 2: [34, 0.17], 3: [32, 0.16], 4: [25, 0.13], 5: [33, 0.17], 6: [38, 0.19]}

196: {1: [34, 0.17], 2: [34, 0.17], 3: [32, 0.16], 4: [25, 0.13], 5: [33, 0.17], 6: [38, 0.19]}

197: {1: [35, 0.18], 2: [34, 0.17], 3: [32, 0.16], 4: [25, 0.13], 5: [33, 0.17], 6: [38, 0.19]}

198: {1: [35, 0.18], 2: [34, 0.17], 3: [33, 0.17], 4: [25, 0.13], 5: [33, 0.17], 6: [38, 0.19]}

199: {1: [35, 0.18], 2: [34, 0.17], 3: [34, 0.17], 4: [25, 0.13], 5: [33, 0.17], 6: [38, 0.19]}

200: {1: [35, 0.17], 2: [35, 0.17], 3: [34, 0.17], 4: [25, 0.12], 5: [33, 0.17], 6: [38, 0.19]}

201: {1: [36, 0.18], 2: [35, 0.17], 3: [34, 0.17], 4: [25, 0.12], 5: [33, 0.16], 6: [38, 0.19]}

202: {1: [36, 0.18], 2: [36, 0.18], 3: [34, 0.17], 4: [25, 0.12], 5: [33, 0.16], 6: [38, 0.19]}

203: {1: [36, 0.18], 2: [36, 0.18], 3: [34, 0.17], 4: [25, 0.12], 5: [34, 0.17], 6: [38, 0.19]}

204: {1: [37, 0.18], 2: [36, 0.18], 3: [34, 0.17], 4: [25, 0.12], 5: [34, 0.17], 6: [38, 0.19]}

205: {1: [38, 0.19], 2: [36, 0.18], 3: [34, 0.17], 4: [25, 0.12], 5: [34, 0.17], 6: [38, 0.19]}

206: {1: [38, 0.18], 2: [36, 0.17], 3: [34, 0.17], 4: [25, 0.12], 5: [34, 0.17], 6: [39, 0.19]}

207: {1: [38, 0.18], 2: [36, 0.17], 3: [34, 0.16], 4: [26, 0.13], 5: [34, 0.16], 6: [39, 0.19]}

208: {1: [39, 0.19], 2: [36, 0.17], 3: [34, 0.16], 4: [26, 0.12], 5: [34, 0.16], 6: [39, 0.19]}

209: {1: [39, 0.19], 2: [36, 0.17], 3: [34, 0.16], 4: [26, 0.12], 5: [35, 0.17], 6: [39, 0.19]}

210: {1: [39, 0.19], 2: [37, 0.18], 3: [34, 0.16], 4: [26, 0.12], 5: [35, 0.17], 6: [39, 0.19]}

211: {1: [39, 0.18], 2: [38, 0.18], 3: [34, 0.16], 4: [26, 0.12], 5: [35, 0.17], 6: [39, 0.18]}

212: {1: [39, 0.18], 2: [38, 0.18], 3: [34, 0.16], 4: [26, 0.12], 5: [35, 0.17], 6: [40, 0.19]}

213: {1: [39, 0.18], 2: [38, 0.18], 3: [34, 0.16], 4: [27, 0.13], 5: [35, 0.16], 6: [40, 0.19]}

214: {1: [39, 0.18], 2: [38, 0.18], 3: [34, 0.16], 4: [27, 0.13], 5: [36, 0.17], 6: [40, 0.19]}

215: {1: [39, 0.18], 2: [39, 0.18], 3: [34, 0.16], 4: [27, 0.13], 5: [36, 0.17], 6: [40, 0.19]}

216: {1: [39, 0.18], 2: [39, 0.18], 3: [35, 0.16], 4: [27, 0.12], 5: [36, 0.17], 6: [40, 0.19]}

217: {1: [39, 0.18], 2: [39, 0.18], 3: [35, 0.16], 4: [27, 0.12], 5: [37, 0.17], 6: [40, 0.18]}

218: {1: [39, 0.18], 2: [39, 0.18], 3: [35, 0.16], 4: [27, 0.12], 5: [38, 0.17], 6: [40, 0.18]}

219: {1: [39, 0.18], 2: [39, 0.18], 3: [35, 0.16], 4: [28, 0.13], 5: [38, 0.17], 6: [40, 0.18]}

220: {1: [39, 0.18], 2: [40, 0.18], 3: [35, 0.16], 4: [28, 0.13], 5: [38, 0.17], 6: [40, 0.18]}

221: {1: [39, 0.18], 2: [40, 0.18], 3: [35, 0.16], 4: [28, 0.13], 5: [39, 0.18], 6: [40, 0.18]}

222: {1: [39, 0.18], 2: [40, 0.18], 3: [35, 0.16], 4: [28, 0.13], 5: [39, 0.18], 6: [41, 0.18]}

223: {1: [39, 0.17], 2: [40, 0.18], 3: [35, 0.16], 4: [28, 0.13], 5: [39, 0.17], 6: [42, 0.19]}

224: {1: [39, 0.17], 2: [41, 0.18], 3: [35, 0.16], 4: [28, 0.12], 5: [39, 0.17], 6: [42, 0.19]}

225: {1: [39, 0.17], 2: [41, 0.18], 3: [35, 0.16], 4: [28, 0.12], 5: [39, 0.17], 6: [43, 0.19]}

226: {1: [39, 0.17], 2: [41, 0.18], 3: [36, 0.16], 4: [28, 0.12], 5: [39, 0.17], 6: [43, 0.19]}

227: {1: [39, 0.17], 2: [41, 0.18], 3: [36, 0.16], 4: [29, 0.13], 5: [39, 0.17], 6: [43, 0.19]}

228: {1: [39, 0.17], 2: [41, 0.18], 3: [36, 0.16], 4: [29, 0.13], 5: [39, 0.17], 6: [44, 0.19]}

229: {1: [39, 0.17], 2: [41, 0.18], 3: [36, 0.16], 4: [29, 0.13], 5: [39, 0.17], 6: [45, 0.2]}

230: {1: [39, 0.17], 2: [41, 0.18], 3: [37, 0.16], 4: [29, 0.13], 5: [39, 0.17], 6: [45, 0.2]}

231: {1: [39, 0.17], 2: [41, 0.18], 3: [37, 0.16], 4: [30, 0.13], 5: [39, 0.17], 6: [45, 0.19]}

232: {1: [39, 0.17], 2: [41, 0.18], 3: [37, 0.16], 4: [30, 0.13], 5: [40, 0.17], 6: [45, 0.19]}

233: {1: [39, 0.17], 2: [41, 0.18], 3: [37, 0.16], 4: [31, 0.13], 5: [40, 0.17], 6: [45, 0.19]}

234: {1: [40, 0.17], 2: [41, 0.18], 3: [37, 0.16], 4: [31, 0.13], 5: [40, 0.17], 6: [45, 0.19]}

235: {1: [40, 0.17], 2: [42, 0.18], 3: [37, 0.16], 4: [31, 0.13], 5: [40, 0.17], 6: [45, 0.19]}

236: {1: [40, 0.17], 2: [43, 0.18], 3: [37, 0.16], 4: [31, 0.13], 5: [40, 0.17], 6: [45, 0.19]}

237: {1: [41, 0.17], 2: [43, 0.18], 3: [37, 0.16], 4: [31, 0.13], 5: [40, 0.17], 6: [45, 0.19]}

238: {1: [41, 0.17], 2: [43, 0.18], 3: [38, 0.16], 4: [31, 0.13], 5: [40, 0.17], 6: [45, 0.19]}

239: {1: [42, 0.18], 2: [43, 0.18], 3: [38, 0.16], 4: [31, 0.13], 5: [40, 0.17], 6: [45, 0.19]}

240: {1: [42, 0.17], 2: [43, 0.18], 3: [38, 0.16], 4: [31, 0.13], 5: [41, 0.17], 6: [45, 0.19]}

241: {1: [42, 0.17], 2: [43, 0.18], 3: [38, 0.16], 4: [31, 0.13], 5: [42, 0.17], 6: [45, 0.19]}

242: {1: [42, 0.17], 2: [44, 0.18], 3: [38, 0.16], 4: [31, 0.13], 5: [42, 0.17], 6: [45, 0.19]}

243: {1: [42, 0.17], 2: [44, 0.18], 3: [38, 0.16], 4: [31, 0.13], 5: [43, 0.18], 6: [45, 0.19]}

244: {1: [42, 0.17], 2: [45, 0.18], 3: [38, 0.16], 4: [31, 0.13], 5: [43, 0.18], 6: [45, 0.18]}

245: {1: [43, 0.18], 2: [45, 0.18], 3: [38, 0.16], 4: [31, 0.13], 5: [43, 0.18], 6: [45, 0.18]}

246: {1: [44, 0.18], 2: [45, 0.18], 3: [38, 0.15], 4: [31, 0.13], 5: [43, 0.17], 6: [45, 0.18]}

247: {1: [44, 0.18], 2: [45, 0.18], 3: [38, 0.15], 4: [31, 0.13], 5: [43, 0.17], 6: [46, 0.19]}

248: {1: [44, 0.18], 2: [45, 0.18], 3: [38, 0.15], 4: [31, 0.12], 5: [43, 0.17], 6: [47, 0.19]}

249: {1: [45, 0.18], 2: [45, 0.18], 3: [38, 0.15], 4: [31, 0.12], 5: [43, 0.17], 6: [47, 0.19]}

250: {1: [45, 0.18], 2: [46, 0.18], 3: [38, 0.15], 4: [31, 0.12], 5: [43, 0.17], 6: [47, 0.19]}

251: {1: [46, 0.18], 2: [46, 0.18], 3: [38, 0.15], 4: [31, 0.12], 5: [43, 0.17], 6: [47, 0.19]}

252: {1: [47, 0.19], 2: [46, 0.18], 3: [38, 0.15], 4: [31, 0.12], 5: [43, 0.17], 6: [47, 0.19]}

253: {1: [47, 0.19], 2: [46, 0.18], 3: [39, 0.15], 4: [31, 0.12], 5: [43, 0.17], 6: [47, 0.19]}

254: {1: [48, 0.19], 2: [46, 0.18], 3: [39, 0.15], 4: [31, 0.12], 5: [43, 0.17], 6: [47, 0.19]}

255: {1: [48, 0.19], 2: [46, 0.18], 3: [39, 0.15], 4: [31, 0.12], 5: [44, 0.17], 6: [47, 0.18]}

256: {1: [48, 0.19], 2: [47, 0.18], 3: [39, 0.15], 4: [31, 0.12], 5: [44, 0.17], 6: [47, 0.18]}

257: {1: [48, 0.19], 2: [47, 0.18], 3: [40, 0.16], 4: [31, 0.12], 5: [44, 0.17], 6: [47, 0.18]}

258: {1: [48, 0.19], 2: [47, 0.18], 3: [40, 0.16], 4: [31, 0.12], 5: [44, 0.17], 6: [48, 0.19]}

259: {1: [48, 0.19], 2: [47, 0.18], 3: [40, 0.15], 4: [32, 0.12], 5: [44, 0.17], 6: [48, 0.19]}

260: {1: [48, 0.18], 2: [48, 0.18], 3: [40, 0.15], 4: [32, 0.12], 5: [44, 0.17], 6: [48, 0.18]}

261: {1: [48, 0.18], 2: [48, 0.18], 3: [40, 0.15], 4: [32, 0.12], 5: [45, 0.17], 6: [48, 0.18]}

262: {1: [48, 0.18], 2: [49, 0.19], 3: [40, 0.15], 4: [32, 0.12], 5: [45, 0.17], 6: [48, 0.18]}

263: {1: [48, 0.18], 2: [49, 0.19], 3: [40, 0.15], 4: [32, 0.12], 5: [45, 0.17], 6: [49, 0.19]}

264: {1: [48, 0.18], 2: [49, 0.19], 3: [40, 0.15], 4: [32, 0.12], 5: [46, 0.17], 6: [49, 0.19]}

265: {1: [48, 0.18], 2: [49, 0.18], 3: [40, 0.15], 4: [32, 0.12], 5: [47, 0.18], 6: [49, 0.18]}

266: {1: [48, 0.18], 2: [49, 0.18], 3: [40, 0.15], 4: [33, 0.12], 5: [47, 0.18], 6: [49, 0.18]}

267: {1: [48, 0.18], 2: [50, 0.19], 3: [40, 0.15], 4: [33, 0.12], 5: [47, 0.18], 6: [49, 0.18]}

268: {1: [48, 0.18], 2: [50, 0.19], 3: [40, 0.15], 4: [34, 0.13], 5: [47, 0.18], 6: [49, 0.18]}

269: {1: [48, 0.18], 2: [50, 0.19], 3: [40, 0.15], 4: [35, 0.13], 5: [47, 0.17], 6: [49, 0.18]}

270: {1: [48, 0.18], 2: [51, 0.19], 3: [40, 0.15], 4: [35, 0.13], 5: [47, 0.17], 6: [49, 0.18]}

271: {1: [49, 0.18], 2: [51, 0.19], 3: [40, 0.15], 4: [35, 0.13], 5: [47, 0.17], 6: [49, 0.18]}

272: {1: [50, 0.18], 2: [51, 0.19], 3: [40, 0.15], 4: [35, 0.13], 5: [47, 0.17], 6: [49, 0.18]}

273: {1: [50, 0.18], 2: [51, 0.19], 3: [40, 0.15], 4: [35, 0.13], 5: [47, 0.17], 6: [50, 0.18]}

274: {1: [50, 0.18], 2: [51, 0.19], 3: [40, 0.15], 4: [36, 0.13], 5: [47, 0.17], 6: [50, 0.18]}

275: {1: [50, 0.18], 2: [51, 0.19], 3: [41, 0.15], 4: [36, 0.13], 5: [47, 0.17], 6: [50, 0.18]}

276: {1: [50, 0.18], 2: [51, 0.18], 3: [41, 0.15], 4: [37, 0.13], 5: [47, 0.17], 6: [50, 0.18]}

277: {1: [50, 0.18], 2: [51, 0.18], 3: [41, 0.15], 4: [38, 0.14], 5: [47, 0.17], 6: [50, 0.18]}

278: {1: [50, 0.18], 2: [51, 0.18], 3: [41, 0.15], 4: [39, 0.14], 5: [47, 0.17], 6: [50, 0.18]}

279: {1: [50, 0.18], 2: [51, 0.18], 3: [41, 0.15], 4: [39, 0.14], 5: [47, 0.17], 6: [51, 0.18]}

280: {1: [51, 0.18], 2: [51, 0.18], 3: [41, 0.15], 4: [39, 0.14], 5: [47, 0.17], 6: [51, 0.18]}

281: {1: [51, 0.18], 2: [51, 0.18], 3: [41, 0.15], 4: [39, 0.14], 5: [47, 0.17], 6: [52, 0.19]}

282: {1: [51, 0.18], 2: [51, 0.18], 3: [41, 0.15], 4: [39, 0.14], 5: [47, 0.17], 6: [53, 0.19]}

283: {1: [51, 0.18], 2: [51, 0.18], 3: [41, 0.14], 4: [39, 0.14], 5: [47, 0.17], 6: [54, 0.19]}

284: {1: [52, 0.18], 2: [51, 0.18], 3: [41, 0.14], 4: [39, 0.14], 5: [47, 0.17], 6: [54, 0.19]}

285: {1: [53, 0.19], 2: [51, 0.18], 3: [41, 0.14], 4: [39, 0.14], 5: [47, 0.16], 6: [54, 0.19]}

286: {1: [53, 0.19], 2: [51, 0.18], 3: [41, 0.14], 4: [40, 0.14], 5: [47, 0.16], 6: [54, 0.19]}

287: {1: [53, 0.18], 2: [51, 0.18], 3: [41, 0.14], 4: [40, 0.14], 5: [47, 0.16], 6: [55, 0.19]}

288: {1: [53, 0.18], 2: [51, 0.18], 3: [42, 0.15], 4: [40, 0.14], 5: [47, 0.16], 6: [55, 0.19]}

289: {1: [54, 0.19], 2: [51, 0.18], 3: [42, 0.15], 4: [40, 0.14], 5: [47, 0.16], 6: [55, 0.19]}

290: {1: [54, 0.19], 2: [52, 0.18], 3: [42, 0.14], 4: [40, 0.14], 5: [47, 0.16], 6: [55, 0.19]}

291: {1: [54, 0.19], 2: [53, 0.18], 3: [42, 0.14], 4: [40, 0.14], 5: [47, 0.16], 6: [55, 0.19]}

292: {1: [54, 0.18], 2: [54, 0.18], 3: [42, 0.14], 4: [40, 0.14], 5: [47, 0.16], 6: [55, 0.19]}

293: {1: [54, 0.18], 2: [54, 0.18], 3: [42, 0.14], 4: [40, 0.14], 5: [48, 0.16], 6: [55, 0.19]}

294: {1: [54, 0.18], 2: [54, 0.18], 3: [42, 0.14], 4: [41, 0.14], 5: [48, 0.16], 6: [55, 0.19]}

295: {1: [54, 0.18], 2: [55, 0.19], 3: [42, 0.14], 4: [41, 0.14], 5: [48, 0.16], 6: [55, 0.19]}

296: {1: [54, 0.18], 2: [55, 0.19], 3: [42, 0.14], 4: [42, 0.14], 5: [48, 0.16], 6: [55, 0.19]}

297: {1: [54, 0.18], 2: [56, 0.19], 3: [42, 0.14], 4: [42, 0.14], 5: [48, 0.16], 6: [55, 0.19]}

298: {1: [54, 0.18], 2: [56, 0.19], 3: [43, 0.14], 4: [42, 0.14], 5: [48, 0.16], 6: [55, 0.18]}

299: {1: [54, 0.18], 2: [56, 0.19], 3: [43, 0.14], 4: [43, 0.14], 5: [48, 0.16], 6: [55, 0.18]}

300: {1: [54, 0.18], 2: [57, 0.19], 3: [43, 0.14], 4: [43, 0.14], 5: [48, 0.16], 6: [55, 0.18]}

301: {1: [55, 0.18], 2: [57, 0.19], 3: [43, 0.14], 4: [43, 0.14], 5: [48, 0.16], 6: [55, 0.18]}

302: {1: [55, 0.18], 2: [57, 0.19], 3: [43, 0.14], 4: [44, 0.15], 5: [48, 0.16], 6: [55, 0.18]}

303: {1: [55, 0.18], 2: [57, 0.19], 3: [43, 0.14], 4: [44, 0.15], 5: [49, 0.16], 6: [55, 0.18]}

304: {1: [56, 0.18], 2: [57, 0.19], 3: [43, 0.14], 4: [44, 0.14], 5: [49, 0.16], 6: [55, 0.18]}

305: {1: [57, 0.19], 2: [57, 0.19], 3: [43, 0.14], 4: [44, 0.14], 5: [49, 0.16], 6: [55, 0.18]}

306: {1: [57, 0.19], 2: [57, 0.19], 3: [43, 0.14], 4: [44, 0.14], 5: [49, 0.16], 6: [56, 0.18]}

307: {1: [57, 0.19], 2: [57, 0.19], 3: [43, 0.14], 4: [44, 0.14], 5: [50, 0.16], 6: [56, 0.18]}

308: {1: [58, 0.19], 2: [57, 0.19], 3: [43, 0.14], 4: [44, 0.14], 5: [50, 0.16], 6: [56, 0.18]}

309: {1: [59, 0.19], 2: [57, 0.18], 3: [43, 0.14], 4: [44, 0.14], 5: [50, 0.16], 6: [56, 0.18]}

310: {1: [59, 0.19], 2: [58, 0.19], 3: [43, 0.14], 4: [44, 0.14], 5: [50, 0.16], 6: [56, 0.18]}

311: {1: [59, 0.19], 2: [59, 0.19], 3: [43, 0.14], 4: [44, 0.14], 5: [50, 0.16], 6: [56, 0.18]}

312: {1: [59, 0.19], 2: [59, 0.19], 3: [43, 0.14], 4: [45, 0.14], 5: [50, 0.16], 6: [56, 0.18]}

313: {1: [59, 0.19], 2: [59, 0.19], 3: [43, 0.14], 4: [46, 0.15], 5: [50, 0.16], 6: [56, 0.18]}

314: {1: [59, 0.19], 2: [59, 0.19], 3: [43, 0.14], 4: [47, 0.15], 5: [50, 0.16], 6: [56, 0.18]}

315: {1: [59, 0.19], 2: [60, 0.19], 3: [43, 0.14], 4: [47, 0.15], 5: [50, 0.16], 6: [56, 0.18]}

316: {1: [59, 0.19], 2: [60, 0.19], 3: [43, 0.14], 4: [48, 0.15], 5: [50, 0.16], 6: [56, 0.18]}

317: {1: [60, 0.19], 2: [60, 0.19], 3: [43, 0.14], 4: [48, 0.15], 5: [50, 0.16], 6: [56, 0.18]}

318: {1: [60, 0.19], 2: [61, 0.19], 3: [43, 0.14], 4: [48, 0.15], 5: [50, 0.16], 6: [56, 0.18]}

319: {1: [60, 0.19], 2: [61, 0.19], 3: [43, 0.13], 4: [49, 0.15], 5: [50, 0.16], 6: [56, 0.18]}

320: {1: [61, 0.19], 2: [61, 0.19], 3: [43, 0.13], 4: [49, 0.15], 5: [50, 0.16], 6: [56, 0.17]}

321: {1: [61, 0.19], 2: [61, 0.19], 3: [43, 0.13], 4: [50, 0.16], 5: [50, 0.16], 6: [56, 0.17]}

322: {1: [61, 0.19], 2: [61, 0.19], 3: [44, 0.14], 4: [50, 0.16], 5: [50, 0.16], 6: [56, 0.17]}

323: {1: [61, 0.19], 2: [61, 0.19], 3: [44, 0.14], 4: [51, 0.16], 5: [50, 0.15], 6: [56, 0.17]}

324: {1: [61, 0.19], 2: [61, 0.19], 3: [45, 0.14], 4: [51, 0.16], 5: [50, 0.15], 6: [56, 0.17]}

325: {1: [61, 0.19], 2: [61, 0.19], 3: [45, 0.14], 4: [52, 0.16], 5: [50, 0.15], 6: [56, 0.17]}

326: {1: [61, 0.19], 2: [61, 0.19], 3: [45, 0.14], 4: [52, 0.16], 5: [50, 0.15], 6: [57, 0.17]}

327: {1: [61, 0.19], 2: [61, 0.19], 3: [45, 0.14], 4: [52, 0.16], 5: [50, 0.15], 6: [58, 0.18]}

328: {1: [61, 0.19], 2: [61, 0.19], 3: [45, 0.14], 4: [52, 0.16], 5: [51, 0.16], 6: [58, 0.18]}

329: {1: [61, 0.19], 2: [61, 0.19], 3: [45, 0.14], 4: [52, 0.16], 5: [51, 0.16], 6: [59, 0.18]}

330: {1: [61, 0.18], 2: [61, 0.18], 3: [45, 0.14], 4: [52, 0.16], 5: [51, 0.15], 6: [60, 0.18]}

331: {1: [61, 0.18], 2: [61, 0.18], 3: [45, 0.14], 4: [53, 0.16], 5: [51, 0.15], 6: [60, 0.18]}

332: {1: [61, 0.18], 2: [62, 0.19], 3: [45, 0.14], 4: [53, 0.16], 5: [51, 0.15], 6: [60, 0.18]}

333: {1: [61, 0.18], 2: [63, 0.19], 3: [45, 0.14], 4: [53, 0.16], 5: [51, 0.15], 6: [60, 0.18]}

334: {1: [61, 0.18], 2: [63, 0.19], 3: [46, 0.14], 4: [53, 0.16], 5: [51, 0.15], 6: [60, 0.18]}

335: {1: [61, 0.18], 2: [64, 0.19], 3: [46, 0.14], 4: [53, 0.16], 5: [51, 0.15], 6: [60, 0.18]}

336: {1: [62, 0.18], 2: [64, 0.19], 3: [46, 0.14], 4: [53, 0.16], 5: [51, 0.15], 6: [60, 0.18]}

337: {1: [62, 0.18], 2: [64, 0.19], 3: [46, 0.14], 4: [53, 0.16], 5: [52, 0.15], 6: [60, 0.18]}

338: {1: [62, 0.18], 2: [64, 0.19], 3: [46, 0.14], 4: [53, 0.16], 5: [52, 0.15], 6: [61, 0.18]}

339: {1: [62, 0.18], 2: [64, 0.19], 3: [46, 0.14], 4: [53, 0.16], 5: [53, 0.16], 6: [61, 0.18]}

340: {1: [63, 0.19], 2: [64, 0.19], 3: [46, 0.14], 4: [53, 0.16], 5: [53, 0.16], 6: [61, 0.18]}

341: {1: [63, 0.18], 2: [64, 0.19], 3: [46, 0.13], 4: [53, 0.16], 5: [53, 0.16], 6: [62, 0.18]}

342: {1: [63, 0.18], 2: [64, 0.19], 3: [47, 0.14], 4: [53, 0.15], 5: [53, 0.15], 6: [62, 0.18]}

343: {1: [64, 0.19], 2: [64, 0.19], 3: [47, 0.14], 4: [53, 0.15], 5: [53, 0.15], 6: [62, 0.18]}

344: {1: [65, 0.19], 2: [64, 0.19], 3: [47, 0.14], 4: [53, 0.15], 5: [53, 0.15], 6: [62, 0.18]}

345: {1: [65, 0.19], 2: [64, 0.19], 3: [47, 0.14], 4: [53, 0.15], 5: [54, 0.16], 6: [62, 0.18]}

346: {1: [65, 0.19], 2: [64, 0.18], 3: [47, 0.14], 4: [54, 0.16], 5: [54, 0.16], 6: [62, 0.18]}

347: {1: [65, 0.19], 2: [64, 0.18], 3: [47, 0.14], 4: [54, 0.16], 5: [55, 0.16], 6: [62, 0.18]}

348: {1: [65, 0.19], 2: [64, 0.18], 3: [47, 0.14], 4: [54, 0.16], 5: [56, 0.16], 6: [62, 0.18]}

349: {1: [65, 0.19], 2: [65, 0.19], 3: [47, 0.13], 4: [54, 0.15], 5: [56, 0.16], 6: [62, 0.18]}

350: {1: [66, 0.19], 2: [65, 0.19], 3: [47, 0.13], 4: [54, 0.15], 5: [56, 0.16], 6: [62, 0.18]}

351: {1: [66, 0.19], 2: [65, 0.19], 3: [47, 0.13], 4: [54, 0.15], 5: [57, 0.16], 6: [62, 0.18]}

352: {1: [67, 0.19], 2: [65, 0.18], 3: [47, 0.13], 4: [54, 0.15], 5: [57, 0.16], 6: [62, 0.18]}

353: {1: [67, 0.19], 2: [66, 0.19], 3: [47, 0.13], 4: [54, 0.15], 5: [57, 0.16], 6: [62, 0.18]}

354: {1: [68, 0.19], 2: [66, 0.19], 3: [47, 0.13], 4: [54, 0.15], 5: [57, 0.16], 6: [62, 0.18]}

355: {1: [68, 0.19], 2: [66, 0.19], 3: [47, 0.13], 4: [54, 0.15], 5: [58, 0.16], 6: [62, 0.17]}

356: {1: [69, 0.19], 2: [66, 0.19], 3: [47, 0.13], 4: [54, 0.15], 5: [58, 0.16], 6: [62, 0.17]}

357: {1: [69, 0.19], 2: [66, 0.18], 3: [47, 0.13], 4: [54, 0.15], 5: [58, 0.16], 6: [63, 0.18]}

358: {1: [69, 0.19], 2: [67, 0.19], 3: [47, 0.13], 4: [54, 0.15], 5: [58, 0.16], 6: [63, 0.18]}

359: {1: [69, 0.19], 2: [67, 0.19], 3: [47, 0.13], 4: [55, 0.15], 5: [58, 0.16], 6: [63, 0.18]}

360: {1: [70, 0.19], 2: [67, 0.19], 3: [47, 0.13], 4: [55, 0.15], 5: [58, 0.16], 6: [63, 0.17]}

361: {1: [70, 0.19], 2: [67, 0.19], 3: [47, 0.13], 4: [55, 0.15], 5: [59, 0.16], 6: [63, 0.17]}

362: {1: [70, 0.19], 2: [68, 0.19], 3: [47, 0.13], 4: [55, 0.15], 5: [59, 0.16], 6: [63, 0.17]}

363: {1: [70, 0.19], 2: [68, 0.19], 3: [47, 0.13], 4: [55, 0.15], 5: [60, 0.17], 6: [63, 0.17]}

364: {1: [70, 0.19], 2: [68, 0.19], 3: [47, 0.13], 4: [55, 0.15], 5: [61, 0.17], 6: [63, 0.17]}

365: {1: [71, 0.19], 2: [68, 0.19], 3: [47, 0.13], 4: [55, 0.15], 5: [61, 0.17], 6: [63, 0.17]}

366: {1: [71, 0.19], 2: [68, 0.19], 3: [47, 0.13], 4: [55, 0.15], 5: [62, 0.17], 6: [63, 0.17]}

367: {1: [72, 0.2], 2: [68, 0.19], 3: [47, 0.13], 4: [55, 0.15], 5: [62, 0.17], 6: [63, 0.17]}

368: {1: [72, 0.2], 2: [68, 0.18], 3: [47, 0.13], 4: [56, 0.15], 5: [62, 0.17], 6: [63, 0.17]}

369: {1: [72, 0.2], 2: [68, 0.18], 3: [47, 0.13], 4: [56, 0.15], 5: [62, 0.17], 6: [64, 0.17]}

370: {1: [72, 0.19], 2: [68, 0.18], 3: [47, 0.13], 4: [56, 0.15], 5: [63, 0.17], 6: [64, 0.17]}

371: {1: [72, 0.19], 2: [68, 0.18], 3: [47, 0.13], 4: [56, 0.15], 5: [64, 0.17], 6: [64, 0.17]}

372: {1: [72, 0.19], 2: [68, 0.18], 3: [47, 0.13], 4: [56, 0.15], 5: [65, 0.17], 6: [64, 0.17]}

373: {1: [72, 0.19], 2: [68, 0.18], 3: [48, 0.13], 4: [56, 0.15], 5: [65, 0.17], 6: [64, 0.17]}

374: {1: [72, 0.19], 2: [68, 0.18], 3: [49, 0.13], 4: [56, 0.15], 5: [65, 0.17], 6: [64, 0.17]}

375: {1: [72, 0.19], 2: [69, 0.18], 3: [49, 0.13], 4: [56, 0.15], 5: [65, 0.17], 6: [64, 0.17]}

376: {1: [72, 0.19], 2: [69, 0.18], 3: [49, 0.13], 4: [56, 0.15], 5: [65, 0.17], 6: [65, 0.17]}

377: {1: [72, 0.19], 2: [69, 0.18], 3: [49, 0.13], 4: [56, 0.15], 5: [65, 0.17], 6: [66, 0.18]}

378: {1: [72, 0.19], 2: [69, 0.18], 3: [50, 0.13], 4: [56, 0.15], 5: [65, 0.17], 6: [66, 0.17]}

379: {1: [72, 0.19], 2: [70, 0.18], 3: [50, 0.13], 4: [56, 0.15], 5: [65, 0.17], 6: [66, 0.17]}

380: {1: [72, 0.19], 2: [70, 0.18], 3: [51, 0.13], 4: [56, 0.15], 5: [65, 0.17], 6: [66, 0.17]}

381: {1: [72, 0.19], 2: [70, 0.18], 3: [51, 0.13], 4: [57, 0.15], 5: [65, 0.17], 6: [66, 0.17]}

382: {1: [72, 0.19], 2: [71, 0.19], 3: [51, 0.13], 4: [57, 0.15], 5: [65, 0.17], 6: [66, 0.17]}

383: {1: [72, 0.19], 2: [71, 0.19], 3: [51, 0.13], 4: [57, 0.15], 5: [65, 0.17], 6: [67, 0.17]}

384: {1: [72, 0.19], 2: [71, 0.18], 3: [51, 0.13], 4: [57, 0.15], 5: [65, 0.17], 6: [68, 0.18]}

385: {1: [72, 0.19], 2: [71, 0.18], 3: [52, 0.14], 4: [57, 0.15], 5: [65, 0.17], 6: [68, 0.18]}

386: {1: [72, 0.19], 2: [71, 0.18], 3: [52, 0.13], 4: [58, 0.15], 5: [65, 0.17], 6: [68, 0.18]}

387: {1: [72, 0.19], 2: [71, 0.18], 3: [53, 0.14], 4: [58, 0.15], 5: [65, 0.17], 6: [68, 0.18]}

388: {1: [73, 0.19], 2: [71, 0.18], 3: [53, 0.14], 4: [58, 0.15], 5: [65, 0.17], 6: [68, 0.18]}

389: {1: [74, 0.19], 2: [71, 0.18], 3: [53, 0.14], 4: [58, 0.15], 5: [65, 0.17], 6: [68, 0.17]}

390: {1: [74, 0.19], 2: [71, 0.18], 3: [53, 0.14], 4: [59, 0.15], 5: [65, 0.17], 6: [68, 0.17]}

391: {1: [74, 0.19], 2: [71, 0.18], 3: [53, 0.14], 4: [59, 0.15], 5: [66, 0.17], 6: [68, 0.17]}

392: {1: [74, 0.19], 2: [71, 0.18], 3: [53, 0.14], 4: [59, 0.15], 5: [67, 0.17], 6: [68, 0.17]}

393: {1: [75, 0.19], 2: [71, 0.18], 3: [53, 0.13], 4: [59, 0.15], 5: [67, 0.17], 6: [68, 0.17]}

394: {1: [76, 0.19], 2: [71, 0.18], 3: [53, 0.13], 4: [59, 0.15], 5: [67, 0.17], 6: [68, 0.17]}

395: {1: [76, 0.19], 2: [71, 0.18], 3: [53, 0.13], 4: [59, 0.15], 5: [68, 0.17], 6: [68, 0.17]}

396: {1: [76, 0.19], 2: [72, 0.18], 3: [53, 0.13], 4: [59, 0.15], 5: [68, 0.17], 6: [68, 0.17]}

397: {1: [76, 0.19], 2: [72, 0.18], 3: [53, 0.13], 4: [59, 0.15], 5: [69, 0.17], 6: [68, 0.17]}

398: {1: [76, 0.19], 2: [72, 0.18], 3: [54, 0.14], 4: [59, 0.15], 5: [69, 0.17], 6: [68, 0.17]}

399: {1: [76, 0.19], 2: [73, 0.18], 3: [54, 0.14], 4: [59, 0.15], 5: [69, 0.17], 6: [68, 0.17]}

400: {1: [76, 0.19], 2: [73, 0.18], 3: [55, 0.14], 4: [59, 0.15], 5: [69, 0.17], 6: [68, 0.17]}

401: {1: [77, 0.19], 2: [73, 0.18], 3: [55, 0.14], 4: [59, 0.15], 5: [69, 0.17], 6: [68, 0.17]}

402: {1: [77, 0.19], 2: [74, 0.18], 3: [55, 0.14], 4: [59, 0.15], 5: [69, 0.17], 6: [68, 0.17]}

403: {1: [77, 0.19], 2: [74, 0.18], 3: [56, 0.14], 4: [59, 0.15], 5: [69, 0.17], 6: [68, 0.17]}

404: {1: [77, 0.19], 2: [74, 0.18], 3: [57, 0.14], 4: [59, 0.15], 5: [69, 0.17], 6: [68, 0.17]}

405: {1: [77, 0.19], 2: [75, 0.19], 3: [57, 0.14], 4: [59, 0.15], 5: [69, 0.17], 6: [68, 0.17]}

406: {1: [77, 0.19], 2: [75, 0.18], 3: [57, 0.14], 4: [60, 0.15], 5: [69, 0.17], 6: [68, 0.17]}

407: {1: [77, 0.19], 2: [75, 0.18], 3: [57, 0.14], 4: [60, 0.15], 5: [70, 0.17], 6: [68, 0.17]}

408: {1: [77, 0.19], 2: [75, 0.18], 3: [57, 0.14], 4: [60, 0.15], 5: [70, 0.17], 6: [69, 0.17]}

409: {1: [77, 0.19], 2: [75, 0.18], 3: [58, 0.14], 4: [60, 0.15], 5: [70, 0.17], 6: [69, 0.17]}

410: {1: [77, 0.19], 2: [75, 0.18], 3: [58, 0.14], 4: [60, 0.15], 5: [71, 0.17], 6: [69, 0.17]}

411: {1: [77, 0.19], 2: [75, 0.18], 3: [58, 0.14], 4: [60, 0.15], 5: [71, 0.17], 6: [70, 0.17]}

412: {1: [77, 0.19], 2: [75, 0.18], 3: [58, 0.14], 4: [60, 0.15], 5: [72, 0.17], 6: [70, 0.17]}

413: {1: [78, 0.19], 2: [75, 0.18], 3: [58, 0.14], 4: [60, 0.15], 5: [72, 0.17], 6: [70, 0.17]}

414: {1: [78, 0.19], 2: [75, 0.18], 3: [58, 0.14], 4: [60, 0.14], 5: [72, 0.17], 6: [71, 0.17]}

415: {1: [78, 0.19], 2: [75, 0.18], 3: [58, 0.14], 4: [60, 0.14], 5: [73, 0.18], 6: [71, 0.17]}

416: {1: [78, 0.19], 2: [75, 0.18], 3: [59, 0.14], 4: [60, 0.14], 5: [73, 0.18], 6: [71, 0.17]}

417: {1: [78, 0.19], 2: [75, 0.18], 3: [59, 0.14], 4: [60, 0.14], 5: [73, 0.18], 6: [72, 0.17]}

418: {1: [79, 0.19], 2: [75, 0.18], 3: [59, 0.14], 4: [60, 0.14], 5: [73, 0.17], 6: [72, 0.17]}

419: {1: [79, 0.19], 2: [76, 0.18], 3: [59, 0.14], 4: [60, 0.14], 5: [73, 0.17], 6: [72, 0.17]}

420: {1: [79, 0.19], 2: [76, 0.18], 3: [60, 0.14], 4: [60, 0.14], 5: [73, 0.17], 6: [72, 0.17]}

421: {1: [80, 0.19], 2: [76, 0.18], 3: [60, 0.14], 4: [60, 0.14], 5: [73, 0.17], 6: [72, 0.17]}

422: {1: [81, 0.19], 2: [76, 0.18], 3: [60, 0.14], 4: [60, 0.14], 5: [73, 0.17], 6: [72, 0.17]}

423: {1: [81, 0.19], 2: [76, 0.18], 3: [60, 0.14], 4: [60, 0.14], 5: [73, 0.17], 6: [73, 0.17]}

424: {1: [81, 0.19], 2: [76, 0.18], 3: [60, 0.14], 4: [60, 0.14], 5: [74, 0.17], 6: [73, 0.17]}

425: {1: [81, 0.19], 2: [77, 0.18], 3: [60, 0.14], 4: [60, 0.14], 5: [74, 0.17], 6: [73, 0.17]}

426: {1: [81, 0.19], 2: [77, 0.18], 3: [61, 0.14], 4: [60, 0.14], 5: [74, 0.17], 6: [73, 0.17]}

427: {1: [81, 0.19], 2: [77, 0.18], 3: [62, 0.15], 4: [60, 0.14], 5: [74, 0.17], 6: [73, 0.17]}

428: {1: [81, 0.19], 2: [77, 0.18], 3: [62, 0.14], 4: [60, 0.14], 5: [75, 0.18], 6: [73, 0.17]}

429: {1: [81, 0.19], 2: [78, 0.18], 3: [62, 0.14], 4: [60, 0.14], 5: [75, 0.17], 6: [73, 0.17]}

430: {1: [81, 0.19], 2: [78, 0.18], 3: [62, 0.14], 4: [60, 0.14], 5: [75, 0.17], 6: [74, 0.17]}

431: {1: [81, 0.19], 2: [78, 0.18], 3: [63, 0.15], 4: [60, 0.14], 5: [75, 0.17], 6: [74, 0.17]}

432: {1: [81, 0.19], 2: [79, 0.18], 3: [63, 0.15], 4: [60, 0.14], 5: [75, 0.17], 6: [74, 0.17]}

433: {1: [81, 0.19], 2: [79, 0.18], 3: [63, 0.15], 4: [60, 0.14], 5: [75, 0.17], 6: [75, 0.17]}

434: {1: [81, 0.19], 2: [79, 0.18], 3: [63, 0.15], 4: [60, 0.14], 5: [75, 0.17], 6: [76, 0.18]}

435: {1: [81, 0.19], 2: [79, 0.18], 3: [64, 0.15], 4: [60, 0.14], 5: [75, 0.17], 6: [76, 0.17]}

436: {1: [81, 0.19], 2: [79, 0.18], 3: [64, 0.15], 4: [60, 0.14], 5: [76, 0.17], 6: [76, 0.17]}

437: {1: [81, 0.19], 2: [79, 0.18], 3: [64, 0.15], 4: [61, 0.14], 5: [76, 0.17], 6: [76, 0.17]}

438: {1: [81, 0.18], 2: [79, 0.18], 3: [65, 0.15], 4: [61, 0.14], 5: [76, 0.17], 6: [76, 0.17]}

439: {1: [82, 0.19], 2: [79, 0.18], 3: [65, 0.15], 4: [61, 0.14], 5: [76, 0.17], 6: [76, 0.17]}

440: {1: [83, 0.19], 2: [79, 0.18], 3: [65, 0.15], 4: [61, 0.14], 5: [76, 0.17], 6: [76, 0.17]}

441: {1: [83, 0.19], 2: [79, 0.18], 3: [65, 0.15], 4: [61, 0.14], 5: [76, 0.17], 6: [77, 0.17]}

442: {1: [83, 0.19], 2: [79, 0.18], 3: [65, 0.15], 4: [62, 0.14], 5: [76, 0.17], 6: [77, 0.17]}

443: {1: [83, 0.19], 2: [79, 0.18], 3: [66, 0.15], 4: [62, 0.14], 5: [76, 0.17], 6: [77, 0.17]}

444: {1: [84, 0.19], 2: [79, 0.18], 3: [66, 0.15], 4: [62, 0.14], 5: [76, 0.17], 6: [77, 0.17]}

445: {1: [85, 0.19], 2: [79, 0.18], 3: [66, 0.15], 4: [62, 0.14], 5: [76, 0.17], 6: [77, 0.17]}

446: {1: [85, 0.19], 2: [79, 0.18], 3: [67, 0.15], 4: [62, 0.14], 5: [76, 0.17], 6: [77, 0.17]}

447: {1: [85, 0.19], 2: [80, 0.18], 3: [67, 0.15], 4: [62, 0.14], 5: [76, 0.17], 6: [77, 0.17]}

448: {1: [85, 0.19], 2: [80, 0.18], 3: [67, 0.15], 4: [62, 0.14], 5: [76, 0.17], 6: [78, 0.17]}

449: {1: [85, 0.19], 2: [80, 0.18], 3: [68, 0.15], 4: [62, 0.14], 5: [76, 0.17], 6: [78, 0.17]}

450: {1: [85, 0.19], 2: [81, 0.18], 3: [68, 0.15], 4: [62, 0.14], 5: [76, 0.17], 6: [78, 0.17]}

451: {1: [85, 0.19], 2: [81, 0.18], 3: [68, 0.15], 4: [62, 0.14], 5: [76, 0.17], 6: [79, 0.18]}

452: {1: [85, 0.19], 2: [81, 0.18], 3: [68, 0.15], 4: [63, 0.14], 5: [76, 0.17], 6: [79, 0.17]}

453: {1: [85, 0.19], 2: [81, 0.18], 3: [68, 0.15], 4: [63, 0.14], 5: [77, 0.17], 6: [79, 0.17]}

454: {1: [85, 0.19], 2: [81, 0.18], 3: [68, 0.15], 4: [63, 0.14], 5: [77, 0.17], 6: [80, 0.18]}

455: {1: [85, 0.19], 2: [81, 0.18], 3: [68, 0.15], 4: [64, 0.14], 5: [77, 0.17], 6: [80, 0.18]}

456: {1: [85, 0.19], 2: [81, 0.18], 3: [68, 0.15], 4: [64, 0.14], 5: [78, 0.17], 6: [80, 0.18]}

457: {1: [86, 0.19], 2: [81, 0.18], 3: [68, 0.15], 4: [64, 0.14], 5: [78, 0.17], 6: [80, 0.18]}

458: {1: [87, 0.19], 2: [81, 0.18], 3: [68, 0.15], 4: [64, 0.14], 5: [78, 0.17], 6: [80, 0.17]}

459: {1: [88, 0.19], 2: [81, 0.18], 3: [68, 0.15], 4: [64, 0.14], 5: [78, 0.17], 6: [80, 0.17]}

460: {1: [88, 0.19], 2: [81, 0.18], 3: [68, 0.15], 4: [64, 0.14], 5: [78, 0.17], 6: [81, 0.18]}

461: {1: [88, 0.19], 2: [82, 0.18], 3: [68, 0.15], 4: [64, 0.14], 5: [78, 0.17], 6: [81, 0.18]}

462: {1: [88, 0.19], 2: [82, 0.18], 3: [68, 0.15], 4: [64, 0.14], 5: [79, 0.17], 6: [81, 0.18]}

463: {1: [89, 0.19], 2: [82, 0.18], 3: [68, 0.15], 4: [64, 0.14], 5: [79, 0.17], 6: [81, 0.17]}

464: {1: [89, 0.19], 2: [82, 0.18], 3: [69, 0.15], 4: [64, 0.14], 5: [79, 0.17], 6: [81, 0.17]}

465: {1: [89, 0.19], 2: [82, 0.18], 3: [69, 0.15], 4: [64, 0.14], 5: [80, 0.17], 6: [81, 0.17]}

466: {1: [89, 0.19], 2: [82, 0.18], 3: [69, 0.15], 4: [64, 0.14], 5: [81, 0.17], 6: [81, 0.17]}

467: {1: [89, 0.19], 2: [83, 0.18], 3: [69, 0.15], 4: [64, 0.14], 5: [81, 0.17], 6: [81, 0.17]}

468: {1: [89, 0.19], 2: [83, 0.18], 3: [69, 0.15], 4: [65, 0.14], 5: [81, 0.17], 6: [81, 0.17]}

469: {1: [89, 0.19], 2: [84, 0.18], 3: [69, 0.15], 4: [65, 0.14], 5: [81, 0.17], 6: [81, 0.17]}

470: {1: [90, 0.19], 2: [84, 0.18], 3: [69, 0.15], 4: [65, 0.14], 5: [81, 0.17], 6: [81, 0.17]}

471: {1: [90, 0.19], 2: [84, 0.18], 3: [70, 0.15], 4: [65, 0.14], 5: [81, 0.17], 6: [81, 0.17]}

472: {1: [90, 0.19], 2: [84, 0.18], 3: [71, 0.15], 4: [65, 0.14], 5: [81, 0.17], 6: [81, 0.17]}

473: {1: [91, 0.19], 2: [84, 0.18], 3: [71, 0.15], 4: [65, 0.14], 5: [81, 0.17], 6: [81, 0.17]}

474: {1: [91, 0.19], 2: [84, 0.18], 3: [72, 0.15], 4: [65, 0.14], 5: [81, 0.17], 6: [81, 0.17]}

475: {1: [91, 0.19], 2: [85, 0.18], 3: [72, 0.15], 4: [65, 0.14], 5: [81, 0.17], 6: [81, 0.17]}

476: {1: [91, 0.19], 2: [85, 0.18], 3: [72, 0.15], 4: [65, 0.14], 5: [81, 0.17], 6: [82, 0.17]}

477: {1: [91, 0.19], 2: [86, 0.18], 3: [72, 0.15], 4: [65, 0.14], 5: [81, 0.17], 6: [82, 0.17]}

478: {1: [91, 0.19], 2: [86, 0.18], 3: [72, 0.15], 4: [65, 0.14], 5: [81, 0.17], 6: [83, 0.17]}

479: {1: [92, 0.19], 2: [86, 0.18], 3: [72, 0.15], 4: [65, 0.14], 5: [81, 0.17], 6: [83, 0.17]}

480: {1: [92, 0.19], 2: [86, 0.18], 3: [73, 0.15], 4: [65, 0.14], 5: [81, 0.17], 6: [83, 0.17]}

481: {1: [92, 0.19], 2: [86, 0.18], 3: [73, 0.15], 4: [65, 0.14], 5: [82, 0.17], 6: [83, 0.17]}

482: {1: [92, 0.19], 2: [86, 0.18], 3: [73, 0.15], 4: [66, 0.14], 5: [82, 0.17], 6: [83, 0.17]}

483: {1: [92, 0.19], 2: [86, 0.18], 3: [73, 0.15], 4: [66, 0.14], 5: [83, 0.17], 6: [83, 0.17]}

484: {1: [92, 0.19], 2: [86, 0.18], 3: [73, 0.15], 4: [66, 0.14], 5: [83, 0.17], 6: [84, 0.17]}

485: {1: [92, 0.19], 2: [87, 0.18], 3: [73, 0.15], 4: [66, 0.14], 5: [83, 0.17], 6: [84, 0.17]}

486: {1: [92, 0.19], 2: [88, 0.18], 3: [73, 0.15], 4: [66, 0.14], 5: [83, 0.17], 6: [84, 0.17]}

487: {1: [92, 0.19], 2: [89, 0.18], 3: [73, 0.15], 4: [66, 0.14], 5: [83, 0.17], 6: [84, 0.17]}

488: {1: [92, 0.19], 2: [90, 0.18], 3: [73, 0.15], 4: [66, 0.14], 5: [83, 0.17], 6: [84, 0.17]}

489: {1: [92, 0.19], 2: [90, 0.18], 3: [73, 0.15], 4: [67, 0.14], 5: [83, 0.17], 6: [84, 0.17]}

490: {1: [93, 0.19], 2: [90, 0.18], 3: [73, 0.15], 4: [67, 0.14], 5: [83, 0.17], 6: [84, 0.17]}

491: {1: [93, 0.19], 2: [91, 0.19], 3: [73, 0.15], 4: [67, 0.14], 5: [83, 0.17], 6: [84, 0.17]}

492: {1: [93, 0.19], 2: [91, 0.18], 3: [73, 0.15], 4: [67, 0.14], 5: [83, 0.17], 6: [85, 0.17]}

493: {1: [93, 0.19], 2: [91, 0.18], 3: [74, 0.15], 4: [67, 0.14], 5: [83, 0.17], 6: [85, 0.17]}

494: {1: [93, 0.19], 2: [91, 0.18], 3: [74, 0.15], 4: [68, 0.14], 5: [83, 0.17], 6: [85, 0.17]}

495: {1: [93, 0.19], 2: [91, 0.18], 3: [74, 0.15], 4: [68, 0.14], 5: [83, 0.17], 6: [86, 0.17]}

496: {1: [93, 0.19], 2: [91, 0.18], 3: [74, 0.15], 4: [68, 0.14], 5: [83, 0.17], 6: [87, 0.18]}

497: {1: [93, 0.19], 2: [92, 0.19], 3: [74, 0.15], 4: [68, 0.14], 5: [83, 0.17], 6: [87, 0.18]}

498: {1: [93, 0.19], 2: [92, 0.18], 3: [75, 0.15], 4: [68, 0.14], 5: [83, 0.17], 6: [87, 0.17]}

499: {1: [93, 0.19], 2: [93, 0.19], 3: [75, 0.15], 4: [68, 0.14], 5: [83, 0.17], 6: [87, 0.17]}

500: {1: [93, 0.19], 2: [93, 0.19], 3: [75, 0.15], 4: [68, 0.14], 5: [83, 0.17], 6: [88, 0.18]}

501: {1: [94, 0.19], 2: [93, 0.19], 3: [75, 0.15], 4: [68, 0.14], 5: [83, 0.17], 6: [88, 0.18]}

502: {1: [94, 0.19], 2: [93, 0.19], 3: [75, 0.15], 4: [69, 0.14], 5: [83, 0.17], 6: [88, 0.18]}

503: {1: [95, 0.19], 2: [93, 0.18], 3: [75, 0.15], 4: [69, 0.14], 5: [83, 0.17], 6: [88, 0.17]}

504: {1: [95, 0.19], 2: [93, 0.18], 3: [75, 0.15], 4: [70, 0.14], 5: [83, 0.16], 6: [88, 0.17]}

505: {1: [95, 0.19], 2: [94, 0.19], 3: [75, 0.15], 4: [70, 0.14], 5: [83, 0.16], 6: [88, 0.17]}

506: {1: [95, 0.19], 2: [95, 0.19], 3: [75, 0.15], 4: [70, 0.14], 5: [83, 0.16], 6: [88, 0.17]}

507: {1: [95, 0.19], 2: [95, 0.19], 3: [75, 0.15], 4: [71, 0.14], 5: [83, 0.16], 6: [88, 0.17]}

508: {1: [95, 0.19], 2: [95, 0.19], 3: [76, 0.15], 4: [71, 0.14], 5: [83, 0.16], 6: [88, 0.17]}

509: {1: [95, 0.19], 2: [95, 0.19], 3: [77, 0.15], 4: [71, 0.14], 5: [83, 0.16], 6: [88, 0.17]}

510: {1: [95, 0.19], 2: [96, 0.19], 3: [77, 0.15], 4: [71, 0.14], 5: [83, 0.16], 6: [88, 0.17]}

511: {1: [95, 0.19], 2: [97, 0.19], 3: [77, 0.15], 4: [71, 0.14], 5: [83, 0.16], 6: [88, 0.17]}

512: {1: [96, 0.19], 2: [97, 0.19], 3: [77, 0.15], 4: [71, 0.14], 5: [83, 0.16], 6: [88, 0.17]}

513: {1: [96, 0.19], 2: [97, 0.19], 3: [77, 0.15], 4: [71, 0.14], 5: [83, 0.16], 6: [89, 0.17]}

514: {1: [96, 0.19], 2: [98, 0.19], 3: [77, 0.15], 4: [71, 0.14], 5: [83, 0.16], 6: [89, 0.17]}

515: {1: [96, 0.19], 2: [98, 0.19], 3: [77, 0.15], 4: [72, 0.14], 5: [83, 0.16], 6: [89, 0.17]}

516: {1: [96, 0.19], 2: [98, 0.19], 3: [78, 0.15], 4: [72, 0.14], 5: [83, 0.16], 6: [89, 0.17]}

517: {1: [96, 0.19], 2: [98, 0.19], 3: [79, 0.15], 4: [72, 0.14], 5: [83, 0.16], 6: [89, 0.17]}

518: {1: [97, 0.19], 2: [98, 0.19], 3: [79, 0.15], 4: [72, 0.14], 5: [83, 0.16], 6: [89, 0.17]}

519: {1: [97, 0.19], 2: [98, 0.19], 3: [80, 0.15], 4: [72, 0.14], 5: [83, 0.16], 6: [89, 0.17]}

520: {1: [97, 0.19], 2: [98, 0.19], 3: [81, 0.16], 4: [72, 0.14], 5: [83, 0.16], 6: [89, 0.17]}

521: {1: [97, 0.19], 2: [98, 0.19], 3: [81, 0.16], 4: [72, 0.14], 5: [83, 0.16], 6: [90, 0.17]}

522: {1: [97, 0.19], 2: [98, 0.19], 3: [81, 0.16], 4: [72, 0.14], 5: [84, 0.16], 6: [90, 0.17]}

523: {1: [97, 0.19], 2: [98, 0.19], 3: [81, 0.15], 4: [73, 0.14], 5: [84, 0.16], 6: [90, 0.17]}

524: {1: [97, 0.19], 2: [98, 0.19], 3: [81, 0.15], 4: [73, 0.14], 5: [84, 0.16], 6: [91, 0.17]}

525: {1: [97, 0.18], 2: [98, 0.19], 3: [81, 0.15], 4: [73, 0.14], 5: [85, 0.16], 6: [91, 0.17]}

526: {1: [97, 0.18], 2: [98, 0.19], 3: [82, 0.16], 4: [73, 0.14], 5: [85, 0.16], 6: [91, 0.17]}

527: {1: [98, 0.19], 2: [98, 0.19], 3: [82, 0.16], 4: [73, 0.14], 5: [85, 0.16], 6: [91, 0.17]}

528: {1: [99, 0.19], 2: [98, 0.19], 3: [82, 0.16], 4: [73, 0.14], 5: [85, 0.16], 6: [91, 0.17]}

529: {1: [99, 0.19], 2: [98, 0.19], 3: [83, 0.16], 4: [73, 0.14], 5: [85, 0.16], 6: [91, 0.17]}

530: {1: [99, 0.19], 2: [99, 0.19], 3: [83, 0.16], 4: [73, 0.14], 5: [85, 0.16], 6: [91, 0.17]}

531: {1: [99, 0.19], 2: [99, 0.19], 3: [83, 0.16], 4: [73, 0.14], 5: [86, 0.16], 6: [91, 0.17]}

532: {1: [99, 0.19], 2: [99, 0.19], 3: [84, 0.16], 4: [73, 0.14], 5: [86, 0.16], 6: [91, 0.17]}

533: {1: [100, 0.19], 2: [99, 0.19], 3: [84, 0.16], 4: [73, 0.14], 5: [86, 0.16], 6: [91, 0.17]}

534: {1: [101, 0.19], 2: [99, 0.19], 3: [84, 0.16], 4: [73, 0.14], 5: [86, 0.16], 6: [91, 0.17]}

535: {1: [101, 0.19], 2: [99, 0.19], 3: [84, 0.16], 4: [73, 0.14], 5: [87, 0.16], 6: [91, 0.17]}

536: {1: [101, 0.19], 2: [99, 0.18], 3: [84, 0.16], 4: [74, 0.14], 5: [87, 0.16], 6: [91, 0.17]}

537: {1: [101, 0.19], 2: [99, 0.18], 3: [85, 0.16], 4: [74, 0.14], 5: [87, 0.16], 6: [91, 0.17]}

538: {1: [101, 0.19], 2: [99, 0.18], 3: [85, 0.16], 4: [74, 0.14], 5: [87, 0.16], 6: [92, 0.17]}

539: {1: [101, 0.19], 2: [99, 0.18], 3: [86, 0.16], 4: [74, 0.14], 5: [87, 0.16], 6: [92, 0.17]}

540: {1: [101, 0.19], 2: [99, 0.18], 3: [86, 0.16], 4: [75, 0.14], 5: [87, 0.16], 6: [92, 0.17]}

541: {1: [101, 0.19], 2: [99, 0.18], 3: [86, 0.16], 4: [75, 0.14], 5: [88, 0.16], 6: [92, 0.17]}

542: {1: [101, 0.19], 2: [99, 0.18], 3: [86, 0.16], 4: [75, 0.14], 5: [89, 0.16], 6: [92, 0.17]}

543: {1: [102, 0.19], 2: [99, 0.18], 3: [86, 0.16], 4: [75, 0.14], 5: [89, 0.16], 6: [92, 0.17]}

544: {1: [102, 0.19], 2: [99, 0.18], 3: [86, 0.16], 4: [76, 0.14], 5: [89, 0.16], 6: [92, 0.17]}

545: {1: [102, 0.19], 2: [99, 0.18], 3: [86, 0.16], 4: [76, 0.14], 5: [90, 0.17], 6: [92, 0.17]}

546: {1: [102, 0.19], 2: [100, 0.18], 3: [86, 0.16], 4: [76, 0.14], 5: [90, 0.16], 6: [92, 0.17]}

547: {1: [102, 0.19], 2: [100, 0.18], 3: [87, 0.16], 4: [76, 0.14], 5: [90, 0.16], 6: [92, 0.17]}

548: {1: [103, 0.19], 2: [100, 0.18], 3: [87, 0.16], 4: [76, 0.14], 5: [90, 0.16], 6: [92, 0.17]}

549: {1: [103, 0.19], 2: [100, 0.18], 3: [87, 0.16], 4: [76, 0.14], 5: [90, 0.16], 6: [93, 0.17]}

550: {1: [103, 0.19], 2: [100, 0.18], 3: [87, 0.16], 4: [77, 0.14], 5: [90, 0.16], 6: [93, 0.17]}

551: {1: [104, 0.19], 2: [100, 0.18], 3: [87, 0.16], 4: [77, 0.14], 5: [90, 0.16], 6: [93, 0.17]}

552: {1: [104, 0.19], 2: [100, 0.18], 3: [87, 0.16], 4: [77, 0.14], 5: [91, 0.16], 6: [93, 0.17]}

553: {1: [104, 0.19], 2: [100, 0.18], 3: [87, 0.16], 4: [77, 0.14], 5: [92, 0.17], 6: [93, 0.17]}

554: {1: [104, 0.19], 2: [100, 0.18], 3: [87, 0.16], 4: [77, 0.14], 5: [92, 0.17], 6: [94, 0.17]}

555: {1: [104, 0.19], 2: [100, 0.18], 3: [87, 0.16], 4: [77, 0.14], 5: [92, 0.17], 6: [95, 0.17]}

556: {1: [104, 0.19], 2: [100, 0.18], 3: [87, 0.16], 4: [77, 0.14], 5: [92, 0.17], 6: [96, 0.17]}

557: {1: [104, 0.19], 2: [100, 0.18], 3: [87, 0.16], 4: [77, 0.14], 5: [92, 0.17], 6: [97, 0.17]}

558: {1: [104, 0.19], 2: [100, 0.18], 3: [87, 0.16], 4: [77, 0.14], 5: [92, 0.16], 6: [98, 0.18]}

559: {1: [104, 0.19], 2: [101, 0.18], 3: [87, 0.16], 4: [77, 0.14], 5: [92, 0.16], 6: [98, 0.18]}

560: {1: [104, 0.19], 2: [101, 0.18], 3: [88, 0.16], 4: [77, 0.14], 5: [92, 0.16], 6: [98, 0.17]}

561: {1: [104, 0.19], 2: [101, 0.18], 3: [88, 0.16], 4: [78, 0.14], 5: [92, 0.16], 6: [98, 0.17]}

562: {1: [105, 0.19], 2: [101, 0.18], 3: [88, 0.16], 4: [78, 0.14], 5: [92, 0.16], 6: [98, 0.17]}

563: {1: [105, 0.19], 2: [101, 0.18], 3: [88, 0.16], 4: [78, 0.14], 5: [92, 0.16], 6: [99, 0.18]}

564: {1: [105, 0.19], 2: [101, 0.18], 3: [89, 0.16], 4: [78, 0.14], 5: [92, 0.16], 6: [99, 0.18]}

565: {1: [105, 0.19], 2: [101, 0.18], 3: [89, 0.16], 4: [78, 0.14], 5: [93, 0.16], 6: [99, 0.18]}

566: {1: [105, 0.19], 2: [101, 0.18], 3: [90, 0.16], 4: [78, 0.14], 5: [93, 0.16], 6: [99, 0.17]}

567: {1: [105, 0.19], 2: [101, 0.18], 3: [90, 0.16], 4: [78, 0.14], 5: [93, 0.16], 6: [100, 0.18]}

568: {1: [106, 0.19], 2: [101, 0.18], 3: [90, 0.16], 4: [78, 0.14], 5: [93, 0.16], 6: [100, 0.18]}

569: {1: [106, 0.19], 2: [101, 0.18], 3: [90, 0.16], 4: [78, 0.14], 5: [93, 0.16], 6: [101, 0.18]}

570: {1: [106, 0.19], 2: [101, 0.18], 3: [91, 0.16], 4: [78, 0.14], 5: [93, 0.16], 6: [101, 0.18]}

571: {1: [106, 0.19], 2: [101, 0.18], 3: [91, 0.16], 4: [78, 0.14], 5: [94, 0.16], 6: [101, 0.18]}

572: {1: [106, 0.19], 2: [101, 0.18], 3: [92, 0.16], 4: [78, 0.14], 5: [94, 0.16], 6: [101, 0.18]}

573: {1: [106, 0.18], 2: [101, 0.18], 3: [92, 0.16], 4: [78, 0.14], 5: [94, 0.16], 6: [102, 0.18]}

574: {1: [106, 0.18], 2: [101, 0.18], 3: [92, 0.16], 4: [79, 0.14], 5: [94, 0.16], 6: [102, 0.18]}

575: {1: [106, 0.18], 2: [101, 0.18], 3: [93, 0.16], 4: [79, 0.14], 5: [94, 0.16], 6: [102, 0.18]}

576: {1: [106, 0.18], 2: [101, 0.18], 3: [93, 0.16], 4: [80, 0.14], 5: [94, 0.16], 6: [102, 0.18]}

577: {1: [106, 0.18], 2: [101, 0.18], 3: [93, 0.16], 4: [80, 0.14], 5: [94, 0.16], 6: [103, 0.18]}

578: {1: [106, 0.18], 2: [101, 0.17], 3: [94, 0.16], 4: [80, 0.14], 5: [94, 0.16], 6: [103, 0.18]}

579: {1: [106, 0.18], 2: [101, 0.17], 3: [94, 0.16], 4: [80, 0.14], 5: [95, 0.16], 6: [103, 0.18]}

580: {1: [106, 0.18], 2: [102, 0.18], 3: [94, 0.16], 4: [80, 0.14], 5: [95, 0.16], 6: [103, 0.18]}

581: {1: [106, 0.18], 2: [103, 0.18], 3: [94, 0.16], 4: [80, 0.14], 5: [95, 0.16], 6: [103, 0.18]}

582: {1: [106, 0.18], 2: [103, 0.18], 3: [94, 0.16], 4: [81, 0.14], 5: [95, 0.16], 6: [103, 0.18]}

583: {1: [106, 0.18], 2: [103, 0.18], 3: [94, 0.16], 4: [81, 0.14], 5: [95, 0.16], 6: [104, 0.18]}

584: {1: [106, 0.18], 2: [103, 0.18], 3: [94, 0.16], 4: [82, 0.14], 5: [95, 0.16], 6: [104, 0.18]}

585: {1: [106, 0.18], 2: [103, 0.18], 3: [94, 0.16], 4: [82, 0.14], 5: [95, 0.16], 6: [105, 0.18]}

586: {1: [106, 0.18], 2: [103, 0.18], 3: [94, 0.16], 4: [82, 0.14], 5: [95, 0.16], 6: [106, 0.18]}

587: {1: [106, 0.18], 2: [104, 0.18], 3: [94, 0.16], 4: [82, 0.14], 5: [95, 0.16], 6: [106, 0.18]}

588: {1: [106, 0.18], 2: [104, 0.18], 3: [94, 0.16], 4: [82, 0.14], 5: [96, 0.16], 6: [106, 0.18]}

589: {1: [106, 0.18], 2: [104, 0.18], 3: [94, 0.16], 4: [82, 0.14], 5: [97, 0.16], 6: [106, 0.18]}

590: {1: [106, 0.18], 2: [104, 0.18], 3: [95, 0.16], 4: [82, 0.14], 5: [97, 0.16], 6: [106, 0.18]}

591: {1: [106, 0.18], 2: [104, 0.18], 3: [95, 0.16], 4: [83, 0.14], 5: [97, 0.16], 6: [106, 0.18]}

592: {1: [106, 0.18], 2: [104, 0.18], 3: [95, 0.16], 4: [83, 0.14], 5: [98, 0.17], 6: [106, 0.18]}

593: {1: [107, 0.18], 2: [104, 0.18], 3: [95, 0.16], 4: [83, 0.14], 5: [98, 0.17], 6: [106, 0.18]}

594: {1: [107, 0.18], 2: [104, 0.18], 3: [96, 0.16], 4: [83, 0.14], 5: [98, 0.16], 6: [106, 0.18]}

595: {1: [107, 0.18], 2: [104, 0.17], 3: [97, 0.16], 4: [83, 0.14], 5: [98, 0.16], 6: [106, 0.18]}

596: {1: [107, 0.18], 2: [105, 0.18], 3: [97, 0.16], 4: [83, 0.14], 5: [98, 0.16], 6: [106, 0.18]}

597: {1: [107, 0.18], 2: [105, 0.18], 3: [97, 0.16], 4: [84, 0.14], 5: [98, 0.16], 6: [106, 0.18]}

598: {1: [107, 0.18], 2: [105, 0.18], 3: [97, 0.16], 4: [84, 0.14], 5: [99, 0.17], 6: [106, 0.18]}

599: {1: [107, 0.18], 2: [105, 0.18], 3: [97, 0.16], 4: [84, 0.14], 5: [100, 0.17], 6: [106, 0.18]}

600: {1: [107, 0.18], 2: [105, 0.17], 3: [97, 0.16], 4: [84, 0.14], 5: [100, 0.17], 6: [107, 0.18]}

601: {1: [107, 0.18], 2: [105, 0.17], 3: [98, 0.16], 4: [84, 0.14], 5: [100, 0.17], 6: [107, 0.18]}

602: {1: [107, 0.18], 2: [105, 0.17], 3: [98, 0.16], 4: [85, 0.14], 5: [100, 0.17], 6: [107, 0.18]}

603: {1: [107, 0.18], 2: [105, 0.17], 3: [98, 0.16], 4: [86, 0.14], 5: [100, 0.17], 6: [107, 0.18]}

604: {1: [107, 0.18], 2: [105, 0.17], 3: [98, 0.16], 4: [87, 0.14], 5: [100, 0.17], 6: [107, 0.18]}

605: {1: [107, 0.18], 2: [105, 0.17], 3: [98, 0.16], 4: [87, 0.14], 5: [100, 0.17], 6: [108, 0.18]}

606: {1: [107, 0.18], 2: [106, 0.17], 3: [98, 0.16], 4: [87, 0.14], 5: [100, 0.17], 6: [108, 0.18]}

607: {1: [107, 0.18], 2: [106, 0.17], 3: [98, 0.16], 4: [87, 0.14], 5: [101, 0.17], 6: [108, 0.18]}

608: {1: [107, 0.18], 2: [106, 0.17], 3: [98, 0.16], 4: [88, 0.14], 5: [101, 0.17], 6: [108, 0.18]}

609: {1: [107, 0.18], 2: [106, 0.17], 3: [98, 0.16], 4: [88, 0.14], 5: [101, 0.17], 6: [109, 0.18]}

610: {1: [107, 0.18], 2: [107, 0.18], 3: [98, 0.16], 4: [88, 0.14], 5: [101, 0.17], 6: [109, 0.18]}

611: {1: [107, 0.18], 2: [107, 0.18], 3: [98, 0.16], 4: [88, 0.14], 5: [101, 0.17], 6: [110, 0.18]}

612: {1: [108, 0.18], 2: [107, 0.17], 3: [98, 0.16], 4: [88, 0.14], 5: [101, 0.17], 6: [110, 0.18]}

613: {1: [108, 0.18], 2: [107, 0.17], 3: [99, 0.16], 4: [88, 0.14], 5: [101, 0.16], 6: [110, 0.18]}

614: {1: [108, 0.18], 2: [107, 0.17], 3: [99, 0.16], 4: [88, 0.14], 5: [102, 0.17], 6: [110, 0.18]}

615: {1: [108, 0.18], 2: [107, 0.17], 3: [99, 0.16], 4: [88, 0.14], 5: [102, 0.17], 6: [111, 0.18]}

616: {1: [108, 0.18], 2: [107, 0.17], 3: [99, 0.16], 4: [88, 0.14], 5: [102, 0.17], 6: [112, 0.18]}

617: {1: [108, 0.18], 2: [107, 0.17], 3: [99, 0.16], 4: [88, 0.14], 5: [103, 0.17], 6: [112, 0.18]}

618: {1: [108, 0.17], 2: [107, 0.17], 3: [100, 0.16], 4: [88, 0.14], 5: [103, 0.17], 6: [112, 0.18]}

619: {1: [109, 0.18], 2: [107, 0.17], 3: [100, 0.16], 4: [88, 0.14], 5: [103, 0.17], 6: [112, 0.18]}

620: {1: [109, 0.18], 2: [108, 0.17], 3: [100, 0.16], 4: [88, 0.14], 5: [103, 0.17], 6: [112, 0.18]}

621: {1: [109, 0.18], 2: [108, 0.17], 3: [100, 0.16], 4: [88, 0.14], 5: [103, 0.17], 6: [113, 0.18]}

622: {1: [109, 0.18], 2: [108, 0.17], 3: [101, 0.16], 4: [88, 0.14], 5: [103, 0.17], 6: [113, 0.18]}

623: {1: [109, 0.17], 2: [109, 0.17], 3: [101, 0.16], 4: [88, 0.14], 5: [103, 0.17], 6: [113, 0.18]}

624: {1: [109, 0.17], 2: [110, 0.18], 3: [101, 0.16], 4: [88, 0.14], 5: [103, 0.17], 6: [113, 0.18]}

625: {1: [109, 0.17], 2: [111, 0.18], 3: [101, 0.16], 4: [88, 0.14], 5: [103, 0.16], 6: [113, 0.18]}

626: {1: [110, 0.18], 2: [111, 0.18], 3: [101, 0.16], 4: [88, 0.14], 5: [103, 0.16], 6: [113, 0.18]}

627: {1: [111, 0.18], 2: [111, 0.18], 3: [101, 0.16], 4: [88, 0.14], 5: [103, 0.16], 6: [113, 0.18]}

628: {1: [111, 0.18], 2: [112, 0.18], 3: [101, 0.16], 4: [88, 0.14], 5: [103, 0.16], 6: [113, 0.18]}

629: {1: [111, 0.18], 2: [112, 0.18], 3: [101, 0.16], 4: [89, 0.14], 5: [103, 0.16], 6: [113, 0.18]}

630: {1: [111, 0.18], 2: [112, 0.18], 3: [101, 0.16], 4: [89, 0.14], 5: [104, 0.17], 6: [113, 0.18]}

631: {1: [112, 0.18], 2: [112, 0.18], 3: [101, 0.16], 4: [89, 0.14], 5: [104, 0.16], 6: [113, 0.18]}

632: {1: [112, 0.18], 2: [112, 0.18], 3: [101, 0.16], 4: [90, 0.14], 5: [104, 0.16], 6: [113, 0.18]}

633: {1: [112, 0.18], 2: [112, 0.18], 3: [101, 0.16], 4: [91, 0.14], 5: [104, 0.16], 6: [113, 0.18]}

634: {1: [112, 0.18], 2: [112, 0.18], 3: [101, 0.16], 4: [91, 0.14], 5: [104, 0.16], 6: [114, 0.18]}

635: {1: [112, 0.18], 2: [112, 0.18], 3: [101, 0.16], 4: [91, 0.14], 5: [105, 0.17], 6: [114, 0.18]}

636: {1: [112, 0.18], 2: [113, 0.18], 3: [101, 0.16], 4: [91, 0.14], 5: [105, 0.17], 6: [114, 0.18]}

637: {1: [112, 0.18], 2: [113, 0.18], 3: [101, 0.16], 4: [91, 0.14], 5: [105, 0.16], 6: [115, 0.18]}

638: {1: [112, 0.18], 2: [113, 0.18], 3: [101, 0.16], 4: [91, 0.14], 5: [105, 0.16], 6: [116, 0.18]}

639: {1: [112, 0.18], 2: [113, 0.18], 3: [101, 0.16], 4: [92, 0.14], 5: [105, 0.16], 6: [116, 0.18]}

640: {1: [112, 0.17], 2: [113, 0.18], 3: [101, 0.16], 4: [93, 0.15], 5: [105, 0.16], 6: [116, 0.18]}

641: {1: [112, 0.17], 2: [113, 0.18], 3: [101, 0.16], 4: [94, 0.15], 5: [105, 0.16], 6: [116, 0.18]}

642: {1: [112, 0.17], 2: [114, 0.18], 3: [101, 0.16], 4: [94, 0.15], 5: [105, 0.16], 6: [116, 0.18]}

643: {1: [112, 0.17], 2: [115, 0.18], 3: [101, 0.16], 4: [94, 0.15], 5: [105, 0.16], 6: [116, 0.18]}

644: {1: [112, 0.17], 2: [115, 0.18], 3: [101, 0.16], 4: [94, 0.15], 5: [105, 0.16], 6: [117, 0.18]}

645: {1: [112, 0.17], 2: [115, 0.18], 3: [101, 0.16], 4: [94, 0.15], 5: [105, 0.16], 6: [118, 0.18]}

646: {1: [113, 0.17], 2: [115, 0.18], 3: [101, 0.16], 4: [94, 0.15], 5: [105, 0.16], 6: [118, 0.18]}

647: {1: [114, 0.18], 2: [115, 0.18], 3: [101, 0.16], 4: [94, 0.15], 5: [105, 0.16], 6: [118, 0.18]}

648: {1: [114, 0.18], 2: [115, 0.18], 3: [101, 0.16], 4: [95, 0.15], 5: [105, 0.16], 6: [118, 0.18]}

649: {1: [114, 0.18], 2: [116, 0.18], 3: [101, 0.16], 4: [95, 0.15], 5: [105, 0.16], 6: [118, 0.18]}

650: {1: [114, 0.18], 2: [117, 0.18], 3: [101, 0.16], 4: [95, 0.15], 5: [105, 0.16], 6: [118, 0.18]}

651: {1: [114, 0.18], 2: [117, 0.18], 3: [101, 0.16], 4: [95, 0.15], 5: [105, 0.16], 6: [119, 0.18]}

652: {1: [114, 0.17], 2: [117, 0.18], 3: [101, 0.15], 4: [95, 0.15], 5: [106, 0.16], 6: [119, 0.18]}

653: {1: [114, 0.17], 2: [117, 0.18], 3: [101, 0.15], 4: [96, 0.15], 5: [106, 0.16], 6: [119, 0.18]}

654: {1: [115, 0.18], 2: [117, 0.18], 3: [101, 0.15], 4: [96, 0.15], 5: [106, 0.16], 6: [119, 0.18]}

655: {1: [115, 0.18], 2: [117, 0.18], 3: [101, 0.15], 4: [96, 0.15], 5: [107, 0.16], 6: [119, 0.18]}

656: {1: [115, 0.18], 2: [118, 0.18], 3: [101, 0.15], 4: [96, 0.15], 5: [107, 0.16], 6: [119, 0.18]}

657: {1: [116, 0.18], 2: [118, 0.18], 3: [101, 0.15], 4: [96, 0.15], 5: [107, 0.16], 6: [119, 0.18]}

658: {1: [116, 0.18], 2: [118, 0.18], 3: [101, 0.15], 4: [97, 0.15], 5: [107, 0.16], 6: [119, 0.18]}

659: {1: [116, 0.18], 2: [119, 0.18], 3: [101, 0.15], 4: [97, 0.15], 5: [107, 0.16], 6: [119, 0.18]}

660: {1: [116, 0.18], 2: [119, 0.18], 3: [101, 0.15], 4: [98, 0.15], 5: [107, 0.16], 6: [119, 0.18]}

661: {1: [116, 0.18], 2: [119, 0.18], 3: [101, 0.15], 4: [98, 0.15], 5: [107, 0.16], 6: [120, 0.18]}

662: {1: [116, 0.18], 2: [119, 0.18], 3: [101, 0.15], 4: [98, 0.15], 5: [108, 0.16], 6: [120, 0.18]}

663: {1: [116, 0.17], 2: [119, 0.18], 3: [101, 0.15], 4: [98, 0.15], 5: [109, 0.16], 6: [120, 0.18]}

664: {1: [116, 0.17], 2: [119, 0.18], 3: [101, 0.15], 4: [98, 0.15], 5: [110, 0.17], 6: [120, 0.18]}

665: {1: [116, 0.17], 2: [119, 0.18], 3: [102, 0.15], 4: [98, 0.15], 5: [110, 0.17], 6: [120, 0.18]}

666: {1: [116, 0.17], 2: [119, 0.18], 3: [102, 0.15], 4: [99, 0.15], 5: [110, 0.17], 6: [120, 0.18]}

667: {1: [116, 0.17], 2: [119, 0.18], 3: [102, 0.15], 4: [99, 0.15], 5: [111, 0.17], 6: [120, 0.18]}

668: {1: [116, 0.17], 2: [119, 0.18], 3: [102, 0.15], 4: [99, 0.15], 5: [111, 0.17], 6: [121, 0.18]}

669: {1: [116, 0.17], 2: [119, 0.18], 3: [102, 0.15], 4: [99, 0.15], 5: [112, 0.17], 6: [121, 0.18]}

670: {1: [116, 0.17], 2: [119, 0.18], 3: [102, 0.15], 4: [100, 0.15], 5: [112, 0.17], 6: [121, 0.18]}

671: {1: [116, 0.17], 2: [119, 0.18], 3: [102, 0.15], 4: [100, 0.15], 5: [113, 0.17], 6: [121, 0.18]}

672: {1: [116, 0.17], 2: [119, 0.18], 3: [102, 0.15], 4: [101, 0.15], 5: [113, 0.17], 6: [121, 0.18]}

673: {1: [116, 0.17], 2: [120, 0.18], 3: [102, 0.15], 4: [101, 0.15], 5: [113, 0.17], 6: [121, 0.18]}

674: {1: [116, 0.17], 2: [120, 0.18], 3: [102, 0.15], 4: [101, 0.15], 5: [113, 0.17], 6: [122, 0.18]}

675: {1: [116, 0.17], 2: [120, 0.18], 3: [102, 0.15], 4: [102, 0.15], 5: [113, 0.17], 6: [122, 0.18]}

676: {1: [116, 0.17], 2: [120, 0.18], 3: [102, 0.15], 4: [103, 0.15], 5: [113, 0.17], 6: [122, 0.18]}

677: {1: [116, 0.17], 2: [120, 0.18], 3: [103, 0.15], 4: [103, 0.15], 5: [113, 0.17], 6: [122, 0.18]}

678: {1: [116, 0.17], 2: [121, 0.18], 3: [103, 0.15], 4: [103, 0.15], 5: [113, 0.17], 6: [122, 0.18]}

679: {1: [116, 0.17], 2: [121, 0.18], 3: [103, 0.15], 4: [103, 0.15], 5: [113, 0.17], 6: [123, 0.18]}

680: {1: [116, 0.17], 2: [121, 0.18], 3: [104, 0.15], 4: [103, 0.15], 5: [113, 0.17], 6: [123, 0.18]}

681: {1: [116, 0.17], 2: [121, 0.18], 3: [104, 0.15], 4: [103, 0.15], 5: [114, 0.17], 6: [123, 0.18]}

682: {1: [116, 0.17], 2: [121, 0.18], 3: [104, 0.15], 4: [104, 0.15], 5: [114, 0.17], 6: [123, 0.18]}

683: {1: [116, 0.17], 2: [121, 0.18], 3: [104, 0.15], 4: [104, 0.15], 5: [114, 0.17], 6: [124, 0.18]}

684: {1: [116, 0.17], 2: [122, 0.18], 3: [104, 0.15], 4: [104, 0.15], 5: [114, 0.17], 6: [124, 0.18]}

685: {1: [116, 0.17], 2: [122, 0.18], 3: [105, 0.15], 4: [104, 0.15], 5: [114, 0.17], 6: [124, 0.18]}

686: {1: [116, 0.17], 2: [122, 0.18], 3: [105, 0.15], 4: [105, 0.15], 5: [114, 0.17], 6: [124, 0.18]}

687: {1: [117, 0.17], 2: [122, 0.18], 3: [105, 0.15], 4: [105, 0.15], 5: [114, 0.17], 6: [124, 0.18]}

688: {1: [117, 0.17], 2: [122, 0.18], 3: [105, 0.15], 4: [105, 0.15], 5: [114, 0.17], 6: [125, 0.18]}

689: {1: [117, 0.17], 2: [122, 0.18], 3: [106, 0.15], 4: [105, 0.15], 5: [114, 0.17], 6: [125, 0.18]}

690: {1: [117, 0.17], 2: [123, 0.18], 3: [106, 0.15], 4: [105, 0.15], 5: [114, 0.17], 6: [125, 0.18]}

691: {1: [117, 0.17], 2: [123, 0.18], 3: [107, 0.15], 4: [105, 0.15], 5: [114, 0.16], 6: [125, 0.18]}

692: {1: [117, 0.17], 2: [123, 0.18], 3: [108, 0.16], 4: [105, 0.15], 5: [114, 0.16], 6: [125, 0.18]}

693: {1: [117, 0.17], 2: [123, 0.18], 3: [109, 0.16], 4: [105, 0.15], 5: [114, 0.16], 6: [125, 0.18]}

694: {1: [117, 0.17], 2: [123, 0.18], 3: [109, 0.16], 4: [105, 0.15], 5: [115, 0.17], 6: [125, 0.18]}

695: {1: [118, 0.17], 2: [123, 0.18], 3: [109, 0.16], 4: [105, 0.15], 5: [115, 0.17], 6: [125, 0.18]}

696: {1: [118, 0.17], 2: [124, 0.18], 3: [109, 0.16], 4: [105, 0.15], 5: [115, 0.17], 6: [125, 0.18]}

697: {1: [118, 0.17], 2: [125, 0.18], 3: [109, 0.16], 4: [105, 0.15], 5: [115, 0.16], 6: [125, 0.18]}

698: {1: [118, 0.17], 2: [126, 0.18], 3: [109, 0.16], 4: [105, 0.15], 5: [115, 0.16], 6: [125, 0.18]}

699: {1: [118, 0.17], 2: [126, 0.18], 3: [109, 0.16], 4: [106, 0.15], 5: [115, 0.16], 6: [125, 0.18]}

700: {1: [118, 0.17], 2: [126, 0.18], 3: [109, 0.16], 4: [106, 0.15], 5: [115, 0.16], 6: [126, 0.18]}

701: {1: [118, 0.17], 2: [127, 0.18], 3: [109, 0.16], 4: [106, 0.15], 5: [115, 0.16], 6: [126, 0.18]}

702: {1: [118, 0.17], 2: [127, 0.18], 3: [109, 0.16], 4: [106, 0.15], 5: [115, 0.16], 6: [127, 0.18]}

703: {1: [118, 0.17], 2: [128, 0.18], 3: [109, 0.16], 4: [106, 0.15], 5: [115, 0.16], 6: [127, 0.18]}

704: {1: [119, 0.17], 2: [128, 0.18], 3: [109, 0.15], 4: [106, 0.15], 5: [115, 0.16], 6: [127, 0.18]}

705: {1: [119, 0.17], 2: [128, 0.18], 3: [109, 0.15], 4: [107, 0.15], 5: [115, 0.16], 6: [127, 0.18]}

706: {1: [119, 0.17], 2: [128, 0.18], 3: [109, 0.15], 4: [108, 0.15], 5: [115, 0.16], 6: [127, 0.18]}

707: {1: [119, 0.17], 2: [128, 0.18], 3: [110, 0.16], 4: [108, 0.15], 5: [115, 0.16], 6: [127, 0.18]}

708: {1: [119, 0.17], 2: [128, 0.18], 3: [110, 0.16], 4: [108, 0.15], 5: [116, 0.16], 6: [127, 0.18]}

709: {1: [119, 0.17], 2: [128, 0.18], 3: [110, 0.16], 4: [109, 0.15], 5: [116, 0.16], 6: [127, 0.18]}

710: {1: [119, 0.17], 2: [128, 0.18], 3: [110, 0.15], 4: [110, 0.15], 5: [116, 0.16], 6: [127, 0.18]}

711: {1: [120, 0.17], 2: [128, 0.18], 3: [110, 0.15], 4: [110, 0.15], 5: [116, 0.16], 6: [127, 0.18]}

712: {1: [120, 0.17], 2: [129, 0.18], 3: [110, 0.15], 4: [110, 0.15], 5: [116, 0.16], 6: [127, 0.18]}

713: {1: [120, 0.17], 2: [129, 0.18], 3: [110, 0.15], 4: [111, 0.16], 5: [116, 0.16], 6: [127, 0.18]}

714: {1: [120, 0.17], 2: [129, 0.18], 3: [110, 0.15], 4: [112, 0.16], 5: [116, 0.16], 6: [127, 0.18]}

715: {1: [120, 0.17], 2: [129, 0.18], 3: [110, 0.15], 4: [112, 0.16], 5: [117, 0.16], 6: [127, 0.18]}

716: {1: [120, 0.17], 2: [129, 0.18], 3: [110, 0.15], 4: [112, 0.16], 5: [117, 0.16], 6: [128, 0.18]}

717: {1: [121, 0.17], 2: [129, 0.18], 3: [110, 0.15], 4: [112, 0.16], 5: [117, 0.16], 6: [128, 0.18]}

718: {1: [121, 0.17], 2: [129, 0.18], 3: [110, 0.15], 4: [112, 0.16], 5: [118, 0.16], 6: [128, 0.18]}

719: {1: [121, 0.17], 2: [129, 0.18], 3: [110, 0.15], 4: [113, 0.16], 5: [118, 0.16], 6: [128, 0.18]}

720: {1: [121, 0.17], 2: [129, 0.18], 3: [110, 0.15], 4: [114, 0.16], 5: [118, 0.16], 6: [128, 0.18]}

721: {1: [122, 0.17], 2: [129, 0.18], 3: [110, 0.15], 4: [114, 0.16], 5: [118, 0.16], 6: [128, 0.18]}

722: {1: [122, 0.17], 2: [129, 0.18], 3: [111, 0.15], 4: [114, 0.16], 5: [118, 0.16], 6: [128, 0.18]}

723: {1: [122, 0.17], 2: [129, 0.18], 3: [112, 0.15], 4: [114, 0.16], 5: [118, 0.16], 6: [128, 0.18]}

724: {1: [122, 0.17], 2: [129, 0.18], 3: [112, 0.15], 4: [115, 0.16], 5: [118, 0.16], 6: [128, 0.18]}

725: {1: [122, 0.17], 2: [129, 0.18], 3: [112, 0.15], 4: [116, 0.16], 5: [118, 0.16], 6: [128, 0.18]}

726: {1: [122, 0.17], 2: [129, 0.18], 3: [112, 0.15], 4: [116, 0.16], 5: [119, 0.16], 6: [128, 0.18]}

727: {1: [122, 0.17], 2: [129, 0.18], 3: [112, 0.15], 4: [116, 0.16], 5: [119, 0.16], 6: [129, 0.18]}

728: {1: [122, 0.17], 2: [129, 0.18], 3: [112, 0.15], 4: [116, 0.16], 5: [119, 0.16], 6: [130, 0.18]}

729: {1: [122, 0.17], 2: [129, 0.18], 3: [112, 0.15], 4: [116, 0.16], 5: [120, 0.16], 6: [130, 0.18]}

730: {1: [122, 0.17], 2: [129, 0.18], 3: [112, 0.15], 4: [116, 0.16], 5: [121, 0.17], 6: [130, 0.18]}

731: {1: [122, 0.17], 2: [130, 0.18], 3: [112, 0.15], 4: [116, 0.16], 5: [121, 0.17], 6: [130, 0.18]}

732: {1: [122, 0.17], 2: [130, 0.18], 3: [112, 0.15], 4: [117, 0.16], 5: [121, 0.17], 6: [130, 0.18]}

733: {1: [122, 0.17], 2: [131, 0.18], 3: [112, 0.15], 4: [117, 0.16], 5: [121, 0.17], 6: [130, 0.18]}

734: {1: [122, 0.17], 2: [131, 0.18], 3: [113, 0.15], 4: [117, 0.16], 5: [121, 0.16], 6: [130, 0.18]}

735: {1: [122, 0.17], 2: [131, 0.18], 3: [113, 0.15], 4: [118, 0.16], 5: [121, 0.16], 6: [130, 0.18]}

736: {1: [122, 0.17], 2: [131, 0.18], 3: [113, 0.15], 4: [119, 0.16], 5: [121, 0.16], 6: [130, 0.18]}

737: {1: [123, 0.17], 2: [131, 0.18], 3: [113, 0.15], 4: [119, 0.16], 5: [121, 0.16], 6: [130, 0.18]}

738: {1: [123, 0.17], 2: [131, 0.18], 3: [113, 0.15], 4: [120, 0.16], 5: [121, 0.16], 6: [130, 0.18]}

739: {1: [123, 0.17], 2: [131, 0.18], 3: [114, 0.15], 4: [120, 0.16], 5: [121, 0.16], 6: [130, 0.18]}

740: {1: [123, 0.17], 2: [131, 0.18], 3: [114, 0.15], 4: [120, 0.16], 5: [121, 0.16], 6: [131, 0.18]}

741: {1: [123, 0.17], 2: [131, 0.18], 3: [114, 0.15], 4: [120, 0.16], 5: [121, 0.16], 6: [132, 0.18]}

742: {1: [123, 0.17], 2: [131, 0.18], 3: [114, 0.15], 4: [121, 0.16], 5: [121, 0.16], 6: [132, 0.18]}

743: {1: [123, 0.17], 2: [131, 0.18], 3: [114, 0.15], 4: [121, 0.16], 5: [121, 0.16], 6: [133, 0.18]}

744: {1: [123, 0.17], 2: [132, 0.18], 3: [114, 0.15], 4: [121, 0.16], 5: [121, 0.16], 6: [133, 0.18]}

745: {1: [123, 0.17], 2: [132, 0.18], 3: [114, 0.15], 4: [122, 0.16], 5: [121, 0.16], 6: [133, 0.18]}

746: {1: [123, 0.16], 2: [133, 0.18], 3: [114, 0.15], 4: [122, 0.16], 5: [121, 0.16], 6: [133, 0.18]}

747: {1: [123, 0.16], 2: [133, 0.18], 3: [114, 0.15], 4: [122, 0.16], 5: [122, 0.16], 6: [133, 0.18]}

748: {1: [123, 0.16], 2: [133, 0.18], 3: [114, 0.15], 4: [122, 0.16], 5: [123, 0.16], 6: [133, 0.18]}

749: {1: [124, 0.17], 2: [133, 0.18], 3: [114, 0.15], 4: [122, 0.16], 5: [123, 0.16], 6: [133, 0.18]}

750: {1: [124, 0.17], 2: [133, 0.18], 3: [114, 0.15], 4: [123, 0.16], 5: [123, 0.16], 6: [133, 0.18]}

751: {1: [124, 0.17], 2: [133, 0.18], 3: [114, 0.15], 4: [123, 0.16], 5: [124, 0.17], 6: [133, 0.18]}

752: {1: [124, 0.16], 2: [133, 0.18], 3: [114, 0.15], 4: [123, 0.16], 5: [125, 0.17], 6: [133, 0.18]}

753: {1: [124, 0.16], 2: [133, 0.18], 3: [114, 0.15], 4: [123, 0.16], 5: [125, 0.17], 6: [134, 0.18]}

754: {1: [125, 0.17], 2: [133, 0.18], 3: [114, 0.15], 4: [123, 0.16], 5: [125, 0.17], 6: [134, 0.18]}

755: {1: [126, 0.17], 2: [133, 0.18], 3: [114, 0.15], 4: [123, 0.16], 5: [125, 0.17], 6: [134, 0.18]}

756: {1: [126, 0.17], 2: [133, 0.18], 3: [114, 0.15], 4: [123, 0.16], 5: [125, 0.17], 6: [135, 0.18]}

757: {1: [126, 0.17], 2: [133, 0.18], 3: [114, 0.15], 4: [124, 0.16], 5: [125, 0.17], 6: [135, 0.18]}

758: {1: [126, 0.17], 2: [134, 0.18], 3: [114, 0.15], 4: [124, 0.16], 5: [125, 0.16], 6: [135, 0.18]}

759: {1: [126, 0.17], 2: [134, 0.18], 3: [114, 0.15], 4: [125, 0.16], 5: [125, 0.16], 6: [135, 0.18]}

760: {1: [126, 0.17], 2: [135, 0.18], 3: [114, 0.15], 4: [125, 0.16], 5: [125, 0.16], 6: [135, 0.18]}

761: {1: [127, 0.17], 2: [135, 0.18], 3: [114, 0.15], 4: [125, 0.16], 5: [125, 0.16], 6: [135, 0.18]}

762: {1: [127, 0.17], 2: [135, 0.18], 3: [115, 0.15], 4: [125, 0.16], 5: [125, 0.16], 6: [135, 0.18]}

763: {1: [127, 0.17], 2: [135, 0.18], 3: [115, 0.15], 4: [126, 0.17], 5: [125, 0.16], 6: [135, 0.18]}

764: {1: [127, 0.17], 2: [135, 0.18], 3: [116, 0.15], 4: [126, 0.16], 5: [125, 0.16], 6: [135, 0.18]}

765: {1: [127, 0.17], 2: [136, 0.18], 3: [116, 0.15], 4: [126, 0.16], 5: [125, 0.16], 6: [135, 0.18]}

766: {1: [127, 0.17], 2: [136, 0.18], 3: [116, 0.15], 4: [127, 0.17], 5: [125, 0.16], 6: [135, 0.18]}

767: {1: [127, 0.17], 2: [136, 0.18], 3: [117, 0.15], 4: [127, 0.17], 5: [125, 0.16], 6: [135, 0.18]}

768: {1: [127, 0.17], 2: [136, 0.18], 3: [118, 0.15], 4: [127, 0.17], 5: [125, 0.16], 6: [135, 0.18]}

769: {1: [127, 0.17], 2: [136, 0.18], 3: [118, 0.15], 4: [128, 0.17], 5: [125, 0.16], 6: [135, 0.18]}

770: {1: [127, 0.16], 2: [136, 0.18], 3: [119, 0.15], 4: [128, 0.17], 5: [125, 0.16], 6: [135, 0.18]}

771: {1: [127, 0.16], 2: [136, 0.18], 3: [119, 0.15], 4: [129, 0.17], 5: [125, 0.16], 6: [135, 0.18]}

772: {1: [127, 0.16], 2: [136, 0.18], 3: [120, 0.16], 4: [129, 0.17], 5: [125, 0.16], 6: [135, 0.17]}

773: {1: [128, 0.17], 2: [136, 0.18], 3: [120, 0.16], 4: [129, 0.17], 5: [125, 0.16], 6: [135, 0.17]}

774: {1: [128, 0.17], 2: [136, 0.18], 3: [120, 0.16], 4: [130, 0.17], 5: [125, 0.16], 6: [135, 0.17]}

775: {1: [129, 0.17], 2: [136, 0.18], 3: [120, 0.15], 4: [130, 0.17], 5: [125, 0.16], 6: [135, 0.17]}

776: {1: [129, 0.17], 2: [136, 0.18], 3: [120, 0.15], 4: [130, 0.17], 5: [125, 0.16], 6: [136, 0.18]}

777: {1: [129, 0.17], 2: [136, 0.18], 3: [120, 0.15], 4: [130, 0.17], 5: [126, 0.16], 6: [136, 0.18]}

778: {1: [130, 0.17], 2: [136, 0.17], 3: [120, 0.15], 4: [130, 0.17], 5: [126, 0.16], 6: [136, 0.17]}

779: {1: [130, 0.17], 2: [136, 0.17], 3: [121, 0.16], 4: [130, 0.17], 5: [126, 0.16], 6: [136, 0.17]}

780: {1: [130, 0.17], 2: [137, 0.18], 3: [121, 0.16], 4: [130, 0.17], 5: [126, 0.16], 6: [136, 0.17]}

781: {1: [130, 0.17], 2: [137, 0.18], 3: [121, 0.15], 4: [130, 0.17], 5: [126, 0.16], 6: [137, 0.18]}

782: {1: [131, 0.17], 2: [137, 0.18], 3: [121, 0.15], 4: [130, 0.17], 5: [126, 0.16], 6: [137, 0.18]}

783: {1: [131, 0.17], 2: [137, 0.17], 3: [121, 0.15], 4: [130, 0.17], 5: [126, 0.16], 6: [138, 0.18]}

784: {1: [132, 0.17], 2: [137, 0.17], 3: [121, 0.15], 4: [130, 0.17], 5: [126, 0.16], 6: [138, 0.18]}

785: {1: [133, 0.17], 2: [137, 0.17], 3: [121, 0.15], 4: [130, 0.17], 5: [126, 0.16], 6: [138, 0.18]}

786: {1: [133, 0.17], 2: [138, 0.18], 3: [121, 0.15], 4: [130, 0.17], 5: [126, 0.16], 6: [138, 0.18]}

787: {1: [133, 0.17], 2: [139, 0.18], 3: [121, 0.15], 4: [130, 0.17], 5: [126, 0.16], 6: [138, 0.18]}

788: {1: [134, 0.17], 2: [139, 0.18], 3: [121, 0.15], 4: [130, 0.16], 5: [126, 0.16], 6: [138, 0.18]}

789: {1: [134, 0.17], 2: [139, 0.18], 3: [121, 0.15], 4: [130, 0.16], 5: [127, 0.16], 6: [138, 0.17]}

790: {1: [134, 0.17], 2: [140, 0.18], 3: [121, 0.15], 4: [130, 0.16], 5: [127, 0.16], 6: [138, 0.17]}

791: {1: [134, 0.17], 2: [141, 0.18], 3: [121, 0.15], 4: [130, 0.16], 5: [127, 0.16], 6: [138, 0.17]}

792: {1: [134, 0.17], 2: [142, 0.18], 3: [121, 0.15], 4: [130, 0.16], 5: [127, 0.16], 6: [138, 0.17]}

793: {1: [134, 0.17], 2: [142, 0.18], 3: [121, 0.15], 4: [131, 0.17], 5: [127, 0.16], 6: [138, 0.17]}

794: {1: [134, 0.17], 2: [142, 0.18], 3: [121, 0.15], 4: [131, 0.16], 5: [127, 0.16], 6: [139, 0.18]}

795: {1: [135, 0.17], 2: [142, 0.18], 3: [121, 0.15], 4: [131, 0.16], 5: [127, 0.16], 6: [139, 0.17]}

796: {1: [135, 0.17], 2: [142, 0.18], 3: [121, 0.15], 4: [131, 0.16], 5: [128, 0.16], 6: [139, 0.17]}

797: {1: [135, 0.17], 2: [143, 0.18], 3: [121, 0.15], 4: [131, 0.16], 5: [128, 0.16], 6: [139, 0.17]}

798: {1: [135, 0.17], 2: [143, 0.18], 3: [121, 0.15], 4: [132, 0.17], 5: [128, 0.16], 6: [139, 0.17]}

799: {1: [135, 0.17], 2: [143, 0.18], 3: [121, 0.15], 4: [132, 0.17], 5: [128, 0.16], 6: [140, 0.18]}

800: {1: [135, 0.17], 2: [143, 0.18], 3: [122, 0.15], 4: [132, 0.17], 5: [128, 0.16], 6: [140, 0.17]}

801: {1: [135, 0.17], 2: [143, 0.18], 3: [123, 0.15], 4: [132, 0.16], 5: [128, 0.16], 6: [140, 0.17]}

802: {1: [135, 0.17], 2: [144, 0.18], 3: [123, 0.15], 4: [132, 0.16], 5: [128, 0.16], 6: [140, 0.17]}

803: {1: [135, 0.17], 2: [144, 0.18], 3: [123, 0.15], 4: [132, 0.16], 5: [129, 0.16], 6: [140, 0.17]}

804: {1: [135, 0.17], 2: [144, 0.18], 3: [123, 0.15], 4: [132, 0.16], 5: [130, 0.16], 6: [140, 0.17]}

805: {1: [135, 0.17], 2: [144, 0.18], 3: [123, 0.15], 4: [132, 0.16], 5: [130, 0.16], 6: [141, 0.18]}

806: {1: [135, 0.17], 2: [144, 0.18], 3: [123, 0.15], 4: [132, 0.16], 5: [130, 0.16], 6: [142, 0.18]}

807: {1: [136, 0.17], 2: [144, 0.18], 3: [123, 0.15], 4: [132, 0.16], 5: [130, 0.16], 6: [142, 0.18]}

808: {1: [136, 0.17], 2: [145, 0.18], 3: [123, 0.15], 4: [132, 0.16], 5: [130, 0.16], 6: [142, 0.18]}

809: {1: [136, 0.17], 2: [145, 0.18], 3: [124, 0.15], 4: [132, 0.16], 5: [130, 0.16], 6: [142, 0.18]}

810: {1: [137, 0.17], 2: [145, 0.18], 3: [124, 0.15], 4: [132, 0.16], 5: [130, 0.16], 6: [142, 0.18]}

811: {1: [137, 0.17], 2: [145, 0.18], 3: [124, 0.15], 4: [132, 0.16], 5: [131, 0.16], 6: [142, 0.18]}

812: {1: [138, 0.17], 2: [145, 0.18], 3: [124, 0.15], 4: [132, 0.16], 5: [131, 0.16], 6: [142, 0.17]}

813: {1: [138, 0.17], 2: [145, 0.18], 3: [125, 0.15], 4: [132, 0.16], 5: [131, 0.16], 6: [142, 0.17]}

814: {1: [138, 0.17], 2: [145, 0.18], 3: [125, 0.15], 4: [132, 0.16], 5: [132, 0.16], 6: [142, 0.17]}

815: {1: [138, 0.17], 2: [145, 0.18], 3: [125, 0.15], 4: [132, 0.16], 5: [132, 0.16], 6: [143, 0.18]}

816: {1: [138, 0.17], 2: [145, 0.18], 3: [125, 0.15], 4: [133, 0.16], 5: [132, 0.16], 6: [143, 0.18]}

817: {1: [138, 0.17], 2: [145, 0.18], 3: [125, 0.15], 4: [134, 0.16], 5: [132, 0.16], 6: [143, 0.18]}

818: {1: [138, 0.17], 2: [145, 0.18], 3: [125, 0.15], 4: [134, 0.16], 5: [132, 0.16], 6: [144, 0.18]}

819: {1: [138, 0.17], 2: [146, 0.18], 3: [125, 0.15], 4: [134, 0.16], 5: [132, 0.16], 6: [144, 0.18]}

820: {1: [139, 0.17], 2: [146, 0.18], 3: [125, 0.15], 4: [134, 0.16], 5: [132, 0.16], 6: [144, 0.18]}

821: {1: [139, 0.17], 2: [146, 0.18], 3: [125, 0.15], 4: [134, 0.16], 5: [133, 0.16], 6: [144, 0.18]}

822: {1: [139, 0.17], 2: [146, 0.18], 3: [125, 0.15], 4: [134, 0.16], 5: [133, 0.16], 6: [145, 0.18]}

823: {1: [139, 0.17], 2: [146, 0.18], 3: [125, 0.15], 4: [134, 0.16], 5: [133, 0.16], 6: [146, 0.18]}

824: {1: [139, 0.17], 2: [147, 0.18], 3: [125, 0.15], 4: [134, 0.16], 5: [133, 0.16], 6: [146, 0.18]}

825: {1: [140, 0.17], 2: [147, 0.18], 3: [125, 0.15], 4: [134, 0.16], 5: [133, 0.16], 6: [146, 0.18]}

826: {1: [140, 0.17], 2: [147, 0.18], 3: [125, 0.15], 4: [134, 0.16], 5: [133, 0.16], 6: [147, 0.18]}

827: {1: [140, 0.17], 2: [147, 0.18], 3: [126, 0.15], 4: [134, 0.16], 5: [133, 0.16], 6: [147, 0.18]}

828: {1: [140, 0.17], 2: [147, 0.18], 3: [126, 0.15], 4: [134, 0.16], 5: [133, 0.16], 6: [148, 0.18]}

829: {1: [140, 0.17], 2: [147, 0.18], 3: [126, 0.15], 4: [134, 0.16], 5: [134, 0.16], 6: [148, 0.18]}

830: {1: [141, 0.17], 2: [147, 0.18], 3: [126, 0.15], 4: [134, 0.16], 5: [134, 0.16], 6: [148, 0.18]}

831: {1: [141, 0.17], 2: [147, 0.18], 3: [126, 0.15], 4: [134, 0.16], 5: [135, 0.16], 6: [148, 0.18]}

832: {1: [142, 0.17], 2: [147, 0.18], 3: [126, 0.15], 4: [134, 0.16], 5: [135, 0.16], 6: [148, 0.18]}

833: {1: [142, 0.17], 2: [147, 0.18], 3: [127, 0.15], 4: [134, 0.16], 5: [135, 0.16], 6: [148, 0.18]}

834: {1: [142, 0.17], 2: [147, 0.18], 3: [127, 0.15], 4: [134, 0.16], 5: [136, 0.16], 6: [148, 0.18]}

835: {1: [142, 0.17], 2: [147, 0.18], 3: [127, 0.15], 4: [135, 0.16], 5: [136, 0.16], 6: [148, 0.18]}

836: {1: [142, 0.17], 2: [147, 0.18], 3: [127, 0.15], 4: [135, 0.16], 5: [136, 0.16], 6: [149, 0.18]}

837: {1: [142, 0.17], 2: [147, 0.18], 3: [128, 0.15], 4: [135, 0.16], 5: [136, 0.16], 6: [149, 0.18]}

838: {1: [143, 0.17], 2: [147, 0.18], 3: [128, 0.15], 4: [135, 0.16], 5: [136, 0.16], 6: [149, 0.18]}

839: {1: [143, 0.17], 2: [147, 0.18], 3: [128, 0.15], 4: [135, 0.16], 5: [137, 0.16], 6: [149, 0.18]}

840: {1: [143, 0.17], 2: [147, 0.17], 3: [128, 0.15], 4: [135, 0.16], 5: [137, 0.16], 6: [150, 0.18]}

841: {1: [143, 0.17], 2: [147, 0.17], 3: [129, 0.15], 4: [135, 0.16], 5: [137, 0.16], 6: [150, 0.18]}

842: {1: [144, 0.17], 2: [147, 0.17], 3: [129, 0.15], 4: [135, 0.16], 5: [137, 0.16], 6: [150, 0.18]}

843: {1: [144, 0.17], 2: [147, 0.17], 3: [129, 0.15], 4: [136, 0.16], 5: [137, 0.16], 6: [150, 0.18]}

844: {1: [144, 0.17], 2: [147, 0.17], 3: [129, 0.15], 4: [137, 0.16], 5: [137, 0.16], 6: [150, 0.18]}

845: {1: [145, 0.17], 2: [147, 0.17], 3: [129, 0.15], 4: [137, 0.16], 5: [137, 0.16], 6: [150, 0.18]}

846: {1: [145, 0.17], 2: [147, 0.17], 3: [129, 0.15], 4: [138, 0.16], 5: [137, 0.16], 6: [150, 0.18]}

847: {1: [145, 0.17], 2: [147, 0.17], 3: [130, 0.15], 4: [138, 0.16], 5: [137, 0.16], 6: [150, 0.18]}

848: {1: [145, 0.17], 2: [147, 0.17], 3: [130, 0.15], 4: [138, 0.16], 5: [137, 0.16], 6: [151, 0.18]}

849: {1: [145, 0.17], 2: [147, 0.17], 3: [130, 0.15], 4: [139, 0.16], 5: [137, 0.16], 6: [151, 0.18]}

850: {1: [145, 0.17], 2: [147, 0.17], 3: [130, 0.15], 4: [139, 0.16], 5: [137, 0.16], 6: [152, 0.18]}

851: {1: [145, 0.17], 2: [148, 0.17], 3: [130, 0.15], 4: [139, 0.16], 5: [137, 0.16], 6: [152, 0.18]}

852: {1: [145, 0.17], 2: [148, 0.17], 3: [130, 0.15], 4: [140, 0.16], 5: [137, 0.16], 6: [152, 0.18]}

853: {1: [145, 0.17], 2: [149, 0.17], 3: [130, 0.15], 4: [140, 0.16], 5: [137, 0.16], 6: [152, 0.18]}

854: {1: [145, 0.17], 2: [149, 0.17], 3: [130, 0.15], 4: [140, 0.16], 5: [137, 0.16], 6: [153, 0.18]}

855: {1: [145, 0.17], 2: [149, 0.17], 3: [130, 0.15], 4: [140, 0.16], 5: [138, 0.16], 6: [153, 0.18]}

856: {1: [145, 0.17], 2: [149, 0.17], 3: [130, 0.15], 4: [140, 0.16], 5: [138, 0.16], 6: [154, 0.18]}

857: {1: [145, 0.17], 2: [149, 0.17], 3: [131, 0.15], 4: [140, 0.16], 5: [138, 0.16], 6: [154, 0.18]}

858: {1: [145, 0.17], 2: [149, 0.17], 3: [131, 0.15], 4: [140, 0.16], 5: [139, 0.16], 6: [154, 0.18]}

859: {1: [145, 0.17], 2: [149, 0.17], 3: [131, 0.15], 4: [140, 0.16], 5: [140, 0.16], 6: [154, 0.18]}

860: {1: [145, 0.17], 2: [149, 0.17], 3: [131, 0.15], 4: [141, 0.16], 5: [140, 0.16], 6: [154, 0.18]}

861: {1: [145, 0.17], 2: [149, 0.17], 3: [131, 0.15], 4: [142, 0.16], 5: [140, 0.16], 6: [154, 0.18]}

862: {1: [145, 0.17], 2: [149, 0.17], 3: [131, 0.15], 4: [143, 0.17], 5: [140, 0.16], 6: [154, 0.18]}

863: {1: [145, 0.17], 2: [149, 0.17], 3: [131, 0.15], 4: [143, 0.17], 5: [140, 0.16], 6: [155, 0.18]}

864: {1: [145, 0.17], 2: [149, 0.17], 3: [131, 0.15], 4: [143, 0.17], 5: [141, 0.16], 6: [155, 0.18]}

865: {1: [145, 0.17], 2: [149, 0.17], 3: [132, 0.15], 4: [143, 0.17], 5: [141, 0.16], 6: [155, 0.18]}

866: {1: [145, 0.17], 2: [149, 0.17], 3: [133, 0.15], 4: [143, 0.17], 5: [141, 0.16], 6: [155, 0.18]}

867: {1: [145, 0.17], 2: [150, 0.17], 3: [133, 0.15], 4: [143, 0.16], 5: [141, 0.16], 6: [155, 0.18]}

868: {1: [146, 0.17], 2: [150, 0.17], 3: [133, 0.15], 4: [143, 0.16], 5: [141, 0.16], 6: [155, 0.18]}

869: {1: [146, 0.17], 2: [150, 0.17], 3: [134, 0.15], 4: [143, 0.16], 5: [141, 0.16], 6: [155, 0.18]}

870: {1: [146, 0.17], 2: [150, 0.17], 3: [134, 0.15], 4: [144, 0.17], 5: [141, 0.16], 6: [155, 0.18]}

871: {1: [146, 0.17], 2: [151, 0.17], 3: [134, 0.15], 4: [144, 0.17], 5: [141, 0.16], 6: [155, 0.18]}

872: {1: [146, 0.17], 2: [151, 0.17], 3: [134, 0.15], 4: [145, 0.17], 5: [141, 0.16], 6: [155, 0.18]}

873: {1: [146, 0.17], 2: [151, 0.17], 3: [134, 0.15], 4: [145, 0.17], 5: [142, 0.16], 6: [155, 0.18]}

874: {1: [146, 0.17], 2: [151, 0.17], 3: [134, 0.15], 4: [145, 0.17], 5: [143, 0.16], 6: [155, 0.18]}

875: {1: [146, 0.17], 2: [151, 0.17], 3: [134, 0.15], 4: [145, 0.17], 5: [143, 0.16], 6: [156, 0.18]}

876: {1: [146, 0.17], 2: [151, 0.17], 3: [134, 0.15], 4: [146, 0.17], 5: [143, 0.16], 6: [156, 0.18]}

877: {1: [146, 0.17], 2: [151, 0.17], 3: [135, 0.15], 4: [146, 0.17], 5: [143, 0.16], 6: [156, 0.18]}

878: {1: [147, 0.17], 2: [151, 0.17], 3: [135, 0.15], 4: [146, 0.17], 5: [143, 0.16], 6: [156, 0.18]}

879: {1: [147, 0.17], 2: [151, 0.17], 3: [135, 0.15], 4: [147, 0.17], 5: [143, 0.16], 6: [156, 0.18]}

880: {1: [147, 0.17], 2: [151, 0.17], 3: [136, 0.15], 4: [147, 0.17], 5: [143, 0.16], 6: [156, 0.18]}

881: {1: [147, 0.17], 2: [152, 0.17], 3: [136, 0.15], 4: [147, 0.17], 5: [143, 0.16], 6: [156, 0.18]}

882: {1: [147, 0.17], 2: [152, 0.17], 3: [136, 0.15], 4: [148, 0.17], 5: [143, 0.16], 6: [156, 0.18]}

883: {1: [147, 0.17], 2: [153, 0.17], 3: [136, 0.15], 4: [148, 0.17], 5: [143, 0.16], 6: [156, 0.18]}

884: {1: [147, 0.17], 2: [153, 0.17], 3: [137, 0.15], 4: [148, 0.17], 5: [143, 0.16], 6: [156, 0.18]}

885: {1: [147, 0.17], 2: [153, 0.17], 3: [138, 0.16], 4: [148, 0.17], 5: [143, 0.16], 6: [156, 0.18]}

886: {1: [147, 0.17], 2: [153, 0.17], 3: [139, 0.16], 4: [148, 0.17], 5: [143, 0.16], 6: [156, 0.18]}

887: {1: [147, 0.17], 2: [153, 0.17], 3: [140, 0.16], 4: [148, 0.17], 5: [143, 0.16], 6: [156, 0.18]}

888: {1: [147, 0.17], 2: [153, 0.17], 3: [140, 0.16], 4: [148, 0.17], 5: [144, 0.16], 6: [156, 0.18]}

889: {1: [147, 0.17], 2: [153, 0.17], 3: [140, 0.16], 4: [148, 0.17], 5: [144, 0.16], 6: [157, 0.18]}

890: {1: [147, 0.17], 2: [153, 0.17], 3: [141, 0.16], 4: [148, 0.17], 5: [144, 0.16], 6: [157, 0.18]}

891: {1: [147, 0.16], 2: [153, 0.17], 3: [141, 0.16], 4: [148, 0.17], 5: [145, 0.16], 6: [157, 0.18]}

892: {1: [148, 0.17], 2: [153, 0.17], 3: [141, 0.16], 4: [148, 0.17], 5: [145, 0.16], 6: [157, 0.18]}

893: {1: [148, 0.17], 2: [153, 0.17], 3: [141, 0.16], 4: [148, 0.17], 5: [146, 0.16], 6: [157, 0.18]}

894: {1: [148, 0.17], 2: [154, 0.17], 3: [141, 0.16], 4: [148, 0.17], 5: [146, 0.16], 6: [157, 0.18]}

895: {1: [149, 0.17], 2: [154, 0.17], 3: [141, 0.16], 4: [148, 0.17], 5: [146, 0.16], 6: [157, 0.18]}

896: {1: [149, 0.17], 2: [155, 0.17], 3: [141, 0.16], 4: [148, 0.17], 5: [146, 0.16], 6: [157, 0.18]}

897: {1: [149, 0.17], 2: [155, 0.17], 3: [141, 0.16], 4: [148, 0.16], 5: [146, 0.16], 6: [158, 0.18]}

898: {1: [149, 0.17], 2: [155, 0.17], 3: [141, 0.16], 4: [149, 0.17], 5: [146, 0.16], 6: [158, 0.18]}

899: {1: [149, 0.17], 2: [155, 0.17], 3: [141, 0.16], 4: [150, 0.17], 5: [146, 0.16], 6: [158, 0.18]}

900: {1: [149, 0.17], 2: [155, 0.17], 3: [141, 0.16], 4: [150, 0.17], 5: [147, 0.16], 6: [158, 0.18]}

901: {1: [149, 0.17], 2: [156, 0.17], 3: [141, 0.16], 4: [150, 0.17], 5: [147, 0.16], 6: [158, 0.18]}

902: {1: [149, 0.17], 2: [156, 0.17], 3: [141, 0.16], 4: [150, 0.17], 5: [147, 0.16], 6: [159, 0.18]}

903: {1: [149, 0.17], 2: [156, 0.17], 3: [141, 0.16], 4: [151, 0.17], 5: [147, 0.16], 6: [159, 0.18]}

904: {1: [149, 0.16], 2: [156, 0.17], 3: [141, 0.16], 4: [151, 0.17], 5: [147, 0.16], 6: [160, 0.18]}

905: {1: [149, 0.16], 2: [156, 0.17], 3: [141, 0.16], 4: [151, 0.17], 5: [147, 0.16], 6: [161, 0.18]}

906: {1: [149, 0.16], 2: [156, 0.17], 3: [141, 0.16], 4: [152, 0.17], 5: [147, 0.16], 6: [161, 0.18]}

907: {1: [149, 0.16], 2: [156, 0.17], 3: [141, 0.16], 4: [153, 0.17], 5: [147, 0.16], 6: [161, 0.18]}

908: {1: [150, 0.17], 2: [156, 0.17], 3: [141, 0.16], 4: [153, 0.17], 5: [147, 0.16], 6: [161, 0.18]}

909: {1: [151, 0.17], 2: [156, 0.17], 3: [141, 0.16], 4: [153, 0.17], 5: [147, 0.16], 6: [161, 0.18]}

910: {1: [151, 0.17], 2: [156, 0.17], 3: [142, 0.16], 4: [153, 0.17], 5: [147, 0.16], 6: [161, 0.18]}

911: {1: [151, 0.17], 2: [157, 0.17], 3: [142, 0.16], 4: [153, 0.17], 5: [147, 0.16], 6: [161, 0.18]}

912: {1: [151, 0.17], 2: [157, 0.17], 3: [142, 0.16], 4: [154, 0.17], 5: [147, 0.16], 6: [161, 0.18]}

913: {1: [151, 0.17], 2: [157, 0.17], 3: [142, 0.16], 4: [154, 0.17], 5: [147, 0.16], 6: [162, 0.18]}

914: {1: [151, 0.17], 2: [158, 0.17], 3: [142, 0.16], 4: [154, 0.17], 5: [147, 0.16], 6: [162, 0.18]}

915: {1: [151, 0.17], 2: [158, 0.17], 3: [143, 0.16], 4: [154, 0.17], 5: [147, 0.16], 6: [162, 0.18]}

916: {1: [151, 0.16], 2: [158, 0.17], 3: [143, 0.16], 4: [154, 0.17], 5: [147, 0.16], 6: [163, 0.18]}

917: {1: [151, 0.16], 2: [158, 0.17], 3: [143, 0.16], 4: [154, 0.17], 5: [148, 0.16], 6: [163, 0.18]}

918: {1: [151, 0.16], 2: [158, 0.17], 3: [144, 0.16], 4: [154, 0.17], 5: [148, 0.16], 6: [163, 0.18]}

919: {1: [151, 0.16], 2: [158, 0.17], 3: [144, 0.16], 4: [155, 0.17], 5: [148, 0.16], 6: [163, 0.18]}

920: {1: [151, 0.16], 2: [158, 0.17], 3: [144, 0.16], 4: [155, 0.17], 5: [149, 0.16], 6: [163, 0.18]}

921: {1: [151, 0.16], 2: [158, 0.17], 3: [144, 0.16], 4: [155, 0.17], 5: [150, 0.16], 6: [163, 0.18]}

922: {1: [151, 0.16], 2: [158, 0.17], 3: [145, 0.16], 4: [155, 0.17], 5: [150, 0.16], 6: [163, 0.18]}

923: {1: [151, 0.16], 2: [158, 0.17], 3: [145, 0.16], 4: [156, 0.17], 5: [150, 0.16], 6: [163, 0.18]}

924: {1: [151, 0.16], 2: [158, 0.17], 3: [145, 0.16], 4: [156, 0.17], 5: [150, 0.16], 6: [164, 0.18]}

925: {1: [151, 0.16], 2: [158, 0.17], 3: [145, 0.16], 4: [156, 0.17], 5: [151, 0.16], 6: [164, 0.18]}

926: {1: [151, 0.16], 2: [158, 0.17], 3: [146, 0.16], 4: [156, 0.17], 5: [151, 0.16], 6: [164, 0.18]}

927: {1: [151, 0.16], 2: [158, 0.17], 3: [147, 0.16], 4: [156, 0.17], 5: [151, 0.16], 6: [164, 0.18]}

928: {1: [151, 0.16], 2: [158, 0.17], 3: [147, 0.16], 4: [156, 0.17], 5: [151, 0.16], 6: [165, 0.18]}

929: {1: [151, 0.16], 2: [158, 0.17], 3: [147, 0.16], 4: [157, 0.17], 5: [151, 0.16], 6: [165, 0.18]}

930: {1: [151, 0.16], 2: [158, 0.17], 3: [148, 0.16], 4: [157, 0.17], 5: [151, 0.16], 6: [165, 0.18]}

931: {1: [151, 0.16], 2: [158, 0.17], 3: [149, 0.16], 4: [157, 0.17], 5: [151, 0.16], 6: [165, 0.18]}

932: {1: [151, 0.16], 2: [158, 0.17], 3: [150, 0.16], 4: [157, 0.17], 5: [151, 0.16], 6: [165, 0.18]}

933: {1: [151, 0.16], 2: [159, 0.17], 3: [150, 0.16], 4: [157, 0.17], 5: [151, 0.16], 6: [165, 0.18]}

934: {1: [152, 0.16], 2: [159, 0.17], 3: [150, 0.16], 4: [157, 0.17], 5: [151, 0.16], 6: [165, 0.18]}

935: {1: [152, 0.16], 2: [159, 0.17], 3: [150, 0.16], 4: [157, 0.17], 5: [151, 0.16], 6: [166, 0.18]}

936: {1: [152, 0.16], 2: [160, 0.17], 3: [150, 0.16], 4: [157, 0.17], 5: [151, 0.16], 6: [166, 0.18]}

937: {1: [152, 0.16], 2: [160, 0.17], 3: [151, 0.16], 4: [157, 0.17], 5: [151, 0.16], 6: [166, 0.18]}

938: {1: [153, 0.16], 2: [160, 0.17], 3: [151, 0.16], 4: [157, 0.17], 5: [151, 0.16], 6: [166, 0.18]}

939: {1: [153, 0.16], 2: [160, 0.17], 3: [151, 0.16], 4: [157, 0.17], 5: [151, 0.16], 6: [167, 0.18]}

940: {1: [153, 0.16], 2: [160, 0.17], 3: [151, 0.16], 4: [157, 0.17], 5: [152, 0.16], 6: [167, 0.18]}

941: {1: [153, 0.16], 2: [160, 0.17], 3: [151, 0.16], 4: [157, 0.17], 5: [152, 0.16], 6: [168, 0.18]}

942: {1: [153, 0.16], 2: [161, 0.17], 3: [151, 0.16], 4: [157, 0.17], 5: [152, 0.16], 6: [168, 0.18]}

943: {1: [153, 0.16], 2: [162, 0.17], 3: [151, 0.16], 4: [157, 0.17], 5: [152, 0.16], 6: [168, 0.18]}

944: {1: [153, 0.16], 2: [163, 0.17], 3: [151, 0.16], 4: [157, 0.17], 5: [152, 0.16], 6: [168, 0.18]}

945: {1: [153, 0.16], 2: [163, 0.17], 3: [151, 0.16], 4: [157, 0.17], 5: [152, 0.16], 6: [169, 0.18]}

946: {1: [153, 0.16], 2: [163, 0.17], 3: [151, 0.16], 4: [158, 0.17], 5: [152, 0.16], 6: [169, 0.18]}

947: {1: [153, 0.16], 2: [163, 0.17], 3: [152, 0.16], 4: [158, 0.17], 5: [152, 0.16], 6: [169, 0.18]}

948: {1: [153, 0.16], 2: [163, 0.17], 3: [152, 0.16], 4: [158, 0.17], 5: [152, 0.16], 6: [170, 0.18]}

949: {1: [153, 0.16], 2: [163, 0.17], 3: [152, 0.16], 4: [158, 0.17], 5: [153, 0.16], 6: [170, 0.18]}

950: {1: [153, 0.16], 2: [163, 0.17], 3: [152, 0.16], 4: [158, 0.17], 5: [154, 0.16], 6: [170, 0.18]}

951: {1: [153, 0.16], 2: [163, 0.17], 3: [152, 0.16], 4: [158, 0.17], 5: [155, 0.16], 6: [170, 0.18]}

952: {1: [153, 0.16], 2: [163, 0.17], 3: [153, 0.16], 4: [158, 0.17], 5: [155, 0.16], 6: [170, 0.18]}

953: {1: [154, 0.16], 2: [163, 0.17], 3: [153, 0.16], 4: [158, 0.17], 5: [155, 0.16], 6: [170, 0.18]}

954: {1: [154, 0.16], 2: [164, 0.17], 3: [153, 0.16], 4: [158, 0.17], 5: [155, 0.16], 6: [170, 0.18]}

955: {1: [154, 0.16], 2: [164, 0.17], 3: [154, 0.16], 4: [158, 0.17], 5: [155, 0.16], 6: [170, 0.18]}

956: {1: [154, 0.16], 2: [165, 0.17], 3: [154, 0.16], 4: [158, 0.17], 5: [155, 0.16], 6: [170, 0.18]}

957: {1: [154, 0.16], 2: [165, 0.17], 3: [155, 0.16], 4: [158, 0.17], 5: [155, 0.16], 6: [170, 0.18]}

958: {1: [154, 0.16], 2: [166, 0.17], 3: [155, 0.16], 4: [158, 0.16], 5: [155, 0.16], 6: [170, 0.18]}

959: {1: [154, 0.16], 2: [166, 0.17], 3: [156, 0.16], 4: [158, 0.16], 5: [155, 0.16], 6: [170, 0.18]}

960: {1: [155, 0.16], 2: [166, 0.17], 3: [156, 0.16], 4: [158, 0.16], 5: [155, 0.16], 6: [170, 0.18]}

961: {1: [155, 0.16], 2: [166, 0.17], 3: [156, 0.16], 4: [158, 0.16], 5: [155, 0.16], 6: [171, 0.18]}

962: {1: [155, 0.16], 2: [166, 0.17], 3: [156, 0.16], 4: [158, 0.16], 5: [156, 0.16], 6: [171, 0.18]}

963: {1: [155, 0.16], 2: [167, 0.17], 3: [156, 0.16], 4: [158, 0.16], 5: [156, 0.16], 6: [171, 0.18]}

964: {1: [155, 0.16], 2: [167, 0.17], 3: [157, 0.16], 4: [158, 0.16], 5: [156, 0.16], 6: [171, 0.18]}

965: {1: [156, 0.16], 2: [167, 0.17], 3: [157, 0.16], 4: [158, 0.16], 5: [156, 0.16], 6: [171, 0.18]}

966: {1: [157, 0.16], 2: [167, 0.17], 3: [157, 0.16], 4: [158, 0.16], 5: [156, 0.16], 6: [171, 0.18]}

967: {1: [157, 0.16], 2: [167, 0.17], 3: [157, 0.16], 4: [158, 0.16], 5: [157, 0.16], 6: [171, 0.18]}

968: {1: [157, 0.16], 2: [167, 0.17], 3: [158, 0.16], 4: [158, 0.16], 5: [157, 0.16], 6: [171, 0.18]}

969: {1: [157, 0.16], 2: [167, 0.17], 3: [158, 0.16], 4: [158, 0.16], 5: [157, 0.16], 6: [172, 0.18]}

970: {1: [157, 0.16], 2: [168, 0.17], 3: [158, 0.16], 4: [158, 0.16], 5: [157, 0.16], 6: [172, 0.18]}

971: {1: [157, 0.16], 2: [168, 0.17], 3: [158, 0.16], 4: [158, 0.16], 5: [157, 0.16], 6: [173, 0.18]}

972: {1: [157, 0.16], 2: [168, 0.17], 3: [158, 0.16], 4: [159, 0.16], 5: [157, 0.16], 6: [173, 0.18]}

973: {1: [158, 0.16], 2: [168, 0.17], 3: [158, 0.16], 4: [159, 0.16], 5: [157, 0.16], 6: [173, 0.18]}

974: {1: [159, 0.16], 2: [168, 0.17], 3: [158, 0.16], 4: [159, 0.16], 5: [157, 0.16], 6: [173, 0.18]}

975: {1: [159, 0.16], 2: [168, 0.17], 3: [158, 0.16], 4: [159, 0.16], 5: [158, 0.16], 6: [173, 0.18]}

976: {1: [159, 0.16], 2: [168, 0.17], 3: [159, 0.16], 4: [159, 0.16], 5: [158, 0.16], 6: [173, 0.18]}

977: {1: [159, 0.16], 2: [168, 0.17], 3: [159, 0.16], 4: [160, 0.16], 5: [158, 0.16], 6: [173, 0.18]}

978: {1: [159, 0.16], 2: [168, 0.17], 3: [159, 0.16], 4: [161, 0.16], 5: [158, 0.16], 6: [173, 0.18]}

979: {1: [159, 0.16], 2: [168, 0.17], 3: [159, 0.16], 4: [162, 0.17], 5: [158, 0.16], 6: [173, 0.18]}

980: {1: [159, 0.16], 2: [168, 0.17], 3: [160, 0.16], 4: [162, 0.17], 5: [158, 0.16], 6: [173, 0.18]}

981: {1: [159, 0.16], 2: [169, 0.17], 3: [160, 0.16], 4: [162, 0.17], 5: [158, 0.16], 6: [173, 0.18]}

982: {1: [160, 0.16], 2: [169, 0.17], 3: [160, 0.16], 4: [162, 0.16], 5: [158, 0.16], 6: [173, 0.18]}

983: {1: [160, 0.16], 2: [169, 0.17], 3: [161, 0.16], 4: [162, 0.16], 5: [158, 0.16], 6: [173, 0.18]}

984: {1: [160, 0.16], 2: [169, 0.17], 3: [161, 0.16], 4: [163, 0.17], 5: [158, 0.16], 6: [173, 0.18]}

985: {1: [161, 0.16], 2: [169, 0.17], 3: [161, 0.16], 4: [163, 0.17], 5: [158, 0.16], 6: [173, 0.18]}

986: {1: [162, 0.16], 2: [169, 0.17], 3: [161, 0.16], 4: [163, 0.17], 5: [158, 0.16], 6: [173, 0.18]}

987: {1: [162, 0.16], 2: [169, 0.17], 3: [161, 0.16], 4: [164, 0.17], 5: [158, 0.16], 6: [173, 0.18]}

988: {1: [162, 0.16], 2: [169, 0.17], 3: [161, 0.16], 4: [165, 0.17], 5: [158, 0.16], 6: [173, 0.18]}

989: {1: [163, 0.16], 2: [169, 0.17], 3: [161, 0.16], 4: [165, 0.17], 5: [158, 0.16], 6: [173, 0.17]}

990: {1: [163, 0.16], 2: [169, 0.17], 3: [161, 0.16], 4: [165, 0.17], 5: [159, 0.16], 6: [173, 0.17]}

991: {1: [163, 0.16], 2: [169, 0.17], 3: [161, 0.16], 4: [165, 0.17], 5: [159, 0.16], 6: [174, 0.18]}

992: {1: [163, 0.16], 2: [169, 0.17], 3: [161, 0.16], 4: [165, 0.17], 5: [159, 0.16], 6: [175, 0.18]}

993: {1: [164, 0.17], 2: [169, 0.17], 3: [161, 0.16], 4: [165, 0.17], 5: [159, 0.16], 6: [175, 0.18]}

994: {1: [164, 0.16], 2: [170, 0.17], 3: [161, 0.16], 4: [165, 0.17], 5: [159, 0.16], 6: [175, 0.18]}

995: {1: [164, 0.16], 2: [171, 0.17], 3: [161, 0.16], 4: [165, 0.17], 5: [159, 0.16], 6: [175, 0.18]}

996: {1: [164, 0.16], 2: [171, 0.17], 3: [161, 0.16], 4: [165, 0.17], 5: [160, 0.16], 6: [175, 0.18]}

997: {1: [164, 0.16], 2: [171, 0.17], 3: [162, 0.16], 4: [165, 0.17], 5: [160, 0.16], 6: [175, 0.18]}

998: {1: [165, 0.17], 2: [171, 0.17], 3: [162, 0.16], 4: [165, 0.17], 5: [160, 0.16], 6: [175, 0.18]}

999: {1: [165, 0.17], 2: [172, 0.17], 3: [162, 0.16], 4: [165, 0.17], 5: [160, 0.16], 6: [175, 0.18]}

1000: {1: [166, 0.17], 2: [172, 0.17], 3: [162, 0.16], 4: [165, 0.17], 5: [160, 0.16], 6: [175, 0.17]}

In [1]:

+0 XP

Done it! As you can see from the output, the more you throw a dice, the more the ratios become equal. These values can be seen as probabilities of a single realization.

**Generator comprehensions**

You are given the following generator functions:

def func1(n):

for i in range(0, n):

yield i\*\*2

def func2(n):

for i in range(0, n):

if i%2 == 0:

yield 2\*i

def func3(n, m):

for i in func1(n):

for j in func2(m):

yield ((i, j), i + j)

**Instructions 3/3**

**30 XP**

* Rewrite func1() as a generator comprehension with nn = 10.
* Rewrite func2() as a generator comprehension with nn = 20.

Incorrect Submission

The system is unable to find your generator comprehension. Don't forget to enclose your expression in round () brackets!

* [3](javascript:void(0))

Rewrite func3() as a generator comprehension with nn = 8 and mm = 10.

[**Take Hint (-9 XP)**](javascript:void(0))

**Incorrect Submission**

Did you return a tuple where the first element is a tuple containing i and j, and the second element the sum of i and j, for each i in the output of the call to the first function, for each j in the output of the call to the second function?

# Rewrite func1() as a generator comprehension

gen = func1(10)

for item in zip(gen, func1(10)):

print(item)

# Rewrite func2() as a generator comprehension

gen = (2\*i for i in range(0, 20) if i%2 == 0)

for item in zip(gen, func2(20)):

print(item)

# Rewrite func3() as a generator comprehension

gen = (((i, j), i + j) for i in func1(8) for j in func2(10))

for item in zip(gen, func3(8, 10)):

print(item)

In [1]: func1

Out[1]: <function \_\_main\_\_.func1>

In [2]: func1(10)

Out[2]: <generator object func1 at 0x7f666fa2afc0>

<script.py> output:

(0, 0)

(1, 1)

(4, 4)

(9, 9)

(16, 16)

(25, 25)

(36, 36)

(49, 49)

(64, 64)

(81, 81)

Traceback (most recent call last):

File "script.py", line 2, in <module>

gen = [2\*i for i in range(0, n) if i%2 == 0]

NameError: name 'n' is not defined

<script.py> output:

(0, 0)

(4, 4)

(8, 8)

(12, 12)

(16, 16)

(20, 20)

(24, 24)

(28, 28)

(32, 32)

(36, 36)

<script.py> output:

(0, 0)

(4, 4)

(8, 8)

(12, 12)

(16, 16)

(20, 20)

(24, 24)

(28, 28)

(32, 32)

(36, 36)

<script.py> output:

(((0, 0), 0), ((0, 0), 0))

(((4, 0), 4), ((0, 4), 4))

(((16, 0), 16), ((0, 8), 8))

(((36, 0), 36), ((0, 12), 12))

(((0, 4), 4), ((0, 16), 16))

(((4, 4), 8), ((1, 0), 1))

(((16, 4), 20), ((1, 4), 5))

(((36, 4), 40), ((1, 8), 9))

(((0, 8), 8), ((1, 12), 13))

(((4, 8), 12), ((1, 16), 17))

(((16, 8), 24), ((4, 0), 4))

(((36, 8), 44), ((4, 4), 8))

(((0, 12), 12), ((4, 8), 12))

(((4, 12), 16), ((4, 12), 16))

(((16, 12), 28), ((4, 16), 20))

(((36, 12), 48), ((9, 0), 9))

(((0, 16), 16), ((9, 4), 13))

(((4, 16), 20), ((9, 8), 17))

(((16, 16), 32), ((9, 12), 21))

(((36, 16), 52), ((9, 16), 25))

In [3]:

<script.py> output:

(((0, 0), 0), ((0, 0), 0))

(((0, 4), 4), ((0, 4), 4))

(((0, 8), 8), ((0, 8), 8))

(((0, 12), 12), ((0, 12), 12))

(((0, 16), 16), ((0, 16), 16))

(((4, 0), 4), ((1, 0), 1))

(((4, 4), 8), ((1, 4), 5))

(((4, 8), 12), ((1, 8), 9))

(((4, 12), 16), ((1, 12), 13))

(((4, 16), 20), ((1, 16), 17))

(((16, 0), 16), ((4, 0), 4))

(((16, 4), 20), ((4, 4), 8))

(((16, 8), 24), ((4, 8), 12))

(((16, 12), 28), ((4, 12), 16))

(((16, 16), 32), ((4, 16), 20))

(((36, 0), 36), ((9, 0), 9))

(((36, 4), 40), ((9, 4), 13))

(((36, 8), 44), ((9, 8), 17))

(((36, 12), 48), ((9, 12), 21))

(((36, 16), 52), ((9, 16), 25))

<script.py> output:

(((0, 0), 0), ((0, 0), 0))

(((1, 0), 1), ((0, 4), 4))

(((4, 0), 4), ((0, 8), 8))

(((9, 0), 9), ((0, 12), 12))

(((16, 0), 16), ((0, 16), 16))

(((25, 0), 25), ((1, 0), 1))

(((36, 0), 36), ((1, 4), 5))

(((49, 0), 49), ((1, 8), 9))

(((0, 4), 4), ((1, 12), 13))

(((1, 4), 5), ((1, 16), 17))

(((4, 4), 8), ((4, 0), 4))

(((9, 4), 13), ((4, 4), 8))

(((16, 4), 20), ((4, 8), 12))

(((25, 4), 29), ((4, 12), 16))

(((36, 4), 40), ((4, 16), 20))

(((49, 4), 53), ((9, 0), 9))

(((0, 8), 8), ((9, 4), 13))

(((1, 8), 9), ((9, 8), 17))

(((4, 8), 12), ((9, 12), 21))

(((9, 8), 17), ((9, 16), 25))

(((16, 8), 24), ((16, 0), 16))

(((25, 8), 33), ((16, 4), 20))

(((36, 8), 44), ((16, 8), 24))

(((49, 8), 57), ((16, 12), 28))

(((0, 12), 12), ((16, 16), 32))

(((1, 12), 13), ((25, 0), 25))

(((4, 12), 16), ((25, 4), 29))

(((9, 12), 21), ((25, 8), 33))

(((16, 12), 28), ((25, 12), 37))

(((25, 12), 37), ((25, 16), 41))

(((36, 12), 48), ((36, 0), 36))

(((49, 12), 61), ((36, 4), 40))

(((0, 16), 16), ((36, 8), 44))

(((1, 16), 17), ((36, 12), 48))

(((4, 16), 20), ((36, 16), 52))

(((9, 16), 25), ((49, 0), 49))

(((16, 16), 32), ((49, 4), 53))

(((25, 16), 41), ((49, 8), 57))

(((36, 16), 52), ((49, 12), 61))

(((49, 16), 65), ((49, 16), 65))

<script.py> output:

(((0, 0), 0), ((0, 0), 0))

(((0, 4), 4), ((0, 4), 4))

(((0, 8), 8), ((0, 8), 8))

(((0, 12), 12), ((0, 12), 12))

(((0, 16), 16), ((0, 16), 16))

(((1, 0), 1), ((1, 0), 1))

(((1, 4), 5), ((1, 4), 5))

(((1, 8), 9), ((1, 8), 9))

(((1, 12), 13), ((1, 12), 13))

(((1, 16), 17), ((1, 16), 17))

(((4, 0), 4), ((4, 0), 4))

(((4, 4), 8), ((4, 4), 8))

(((4, 8), 12), ((4, 8), 12))

(((4, 12), 16), ((4, 12), 16))

(((4, 16), 20), ((4, 16), 20))

(((9, 0), 9), ((9, 0), 9))

(((9, 4), 13), ((9, 4), 13))

(((9, 8), 17), ((9, 8), 17))

(((9, 12), 21), ((9, 12), 21))

(((9, 16), 25), ((9, 16), 25))

(((16, 0), 16), ((16, 0), 16))

(((16, 4), 20), ((16, 4), 20))

(((16, 8), 24), ((16, 8), 24))

(((16, 12), 28), ((16, 12), 28))

(((16, 16), 32), ((16, 16), 32))

(((25, 0), 25), ((25, 0), 25))

(((25, 4), 29), ((25, 4), 29))

(((25, 8), 33), ((25, 8), 33))

(((25, 12), 37), ((25, 12), 37))

(((25, 16), 41), ((25, 16), 41))

(((36, 0), 36), ((36, 0), 36))

(((36, 4), 40), ((36, 4), 40))

(((36, 8), 44), ((36, 8), 44))

(((36, 12), 48), ((36, 12), 48))

(((36, 16), 52), ((36, 16), 52))

(((49, 0), 49), ((49, 0), 49))

(((49, 4), 53), ((49, 4), 53))

(((49, 8), 57), ((49, 8), 57))

(((49, 12), 61), ((49, 12), 61))

(((49, 16), 65), ((49, 16), 65))

In [3]:

+100 XP

Nice work! Now you are able to answer any question on generators. Onwards to the next chapter where you will deep dive into functions!

**Exercise**

**Exercise**

**Positional arguments of variable size**

Let's practice positional arguments of variable size. Your task is to define the function sort\_types(). It takes a variable number of positional arguments and checks if each argument is a number or a string. The checked item is inserted afterwards either in the nums or strings list. Eventually, the function returns a tuple containing these lists.

Use the Python's built-in isinstance() function to check if an object is of a certain type (*e.g.* isinstance(1, int) returns True) or one of the types (*e.g.* isinstance(5.65, (int, str)) returns False).

Types to use in this task are int, float, and str.

**Instructions**

**100 XP**

* Define the function with an arbitrary number of arguments.
* Check if arg is a number and add it to nums if necessary.
* Check if arg is a string and add it to strings if necessary.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

nums

# Define the function with an arbitrary number of arguments

def sort\_types(\*args):

nums, strings = [], []

for arg in args:

# Check if 'arg' is a number and add it to 'nums'

if isinstance(arg, (int, float)):

nums += [arg]

# Check if 'arg' is a string and add it to 'strings'

elif isinstance(arg, str):

strings += [arg]

return (nums, strings)

print(sort\_types(1.57, 'car', 'hat', 4, 5, 'tree', 0.89))

Traceback (most recent call last):

File "script.py", line 15, in <module>

print(sort\_types(1.57, 'car', 'hat', 4, 5, 'tree', 0.89))

TypeError: sort\_types() takes 1 positional argument but 7 were given

<script.py> output:

([4, 5], ['car', 'hat', 'tree'])

<script.py> output:

([4, 5], ['car', 'hat', 'tree'])

<script.py> output:

([4, 5], ['car', 'hat', 'tree'])

<script.py> output:

([4, 5], ['car', 'hat', 'tree'])

<script.py> output:

(['car', 'hat', 4, 5, 'tree'], [])

<script.py> output:

([1.57, 4, 5, 0.89], ['car', 'hat', 'tree'])

<script.py> output:

([1.57, 4, 5, 0.89], ['car', 'hat', 'tree'])

<script.py> output:

([1.57, 4, 5, 0.89], ['car', 'hat', 'tree'])

<script.py> output:

([1.57, 4, 5, 0.89], ['car', 'hat', 'tree'])

In [1]:

+0 XP

Good! You can extend this code to sort many more data types.

**Exercise**

**Exercise**

**Keyword arguments of variable size**

Now let's move to keyword arguments of variable size! Your task is to define the function key\_types(). It should take a variable number of keyword arguments and return a new dictionary: the keys are unique object types of arguments passed to the key\_types() function and the associated values represent lists. Each list should contain argument names that follow the type defined as a key (*e.g.* calling the key\_types(val1='a', val2='b', val3=1) results in {<class 'int'>: ['val3'], <class 'str'>: ['val1', 'val2']}).

To retrieve the type of an object, you need to use the type() function (*e.g.* type(1) is int).

**Instructions**

**100 XP**

* Define the function with an arbitrary number of keyword arguments.
* Iterate over key-value pairs.
* Update a list associated with a key.

[**Take Hint (-30 XP)**](javascript:void(0))

[**Show Answer (-70 XP)**](javascript:void(0))

**Hint**

* If a key already exists in the dict\_type, update the associated list.
* If a key does not exists in the dict\_type, create a new list and add an argument name as the first item.

**Incorrect Submission**

Check the first if statement. Did you correctly specify the body? Did you call dict\_type.append()?

Your code generated an error. Fix it and try again!

# Define the function with an arbitrary number of arguments

def key\_types(\*\*kwargs):

dict\_type = dict()

# Iterate over key value pairs

for key, value in kwargs.items():

# Update a list associated with a key

if type(key) in dict\_type:

dict\_type[type(key)] += [key]

#dict\_type[type(key)].append(key)

else:

dict\_type[type(key)] = [key]

return dict\_type

res = key\_types(a=1, b=2, c=(1, 2), d=3.1, e=4.2)

print(res)

<script.py> output:

{<class 'str'>: ['a', 'b', 'c', 'd', 'e']}

<script.py> output:

{<class 'str'>: ['a', 'b', 'c', 'd', 'e']}

In [1]:

+70 XP

Nicely done! Now you practiced both positional and keyword arguments! Keep going!

**Exercise**

**Exercise**

**Combining argument types**

Now you'll try to combine different argument types. Your task is to define the sort\_all\_types() function. It takes positional and keyword arguments of variable size, finds all the numbers and strings contained within them, and concatenates type-wise the results. Use the sort\_types() function you defined before (available in the workspace). It takes a positional argument of variable size and returns a tuple containing a list of numbers and a list of strings (type sort\_types? to get additional help).

Tip: You'll have to recall another usage of the \* symbol.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Define the arguments passed to the function (use any name you want).
* Find all the numbers and strings in the 1st argument.
* Find all the numbers and strings in the 2nd argument.
* Remember that you will need to access the *values* of the keyword arguments when calling the function sort\_types().

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Check the definition of sort\_all\_types(). Did you correctly specify the body? Did you remember to use .values() to access the values of your keyword arguments?

Awesome, thanks for your feedback!

Did you correctly sort the types for both *args and \**kwargs?

# Define the arguments passed to the function

def sort\_all\_types(\*mylist, \*\*mydict):

# Find all the numbers and strings in the 1st argument

nums1, strings1 = sort\_types(\*mylist)

# Find all the numbers and strings in the 2nd argument

nums2, strings2 = sort\_types(\*mydict.values())

return (nums1 + nums2, strings1 + strings2)

res = sort\_all\_types(

1, 2.0, 'dog', 5.1, num1 = 0.0, num2 = 5, str1 = 'cat'

)

print(res)

<script.py> output:

([], [])

<script.py> output:

([], [])

<script.py> output:

([1, 2.0, 5.1, 0.0, 5], ['dog', 'cat'])

In [1]:

+100 XP

Mastered! Arguments of variable size sometimes seem to be very confusing. But now you can handle them very well!

**Exercise**

**Exercise**

**Define lambda expressions**

Let's write some lambda expressions! You will be given three tasks: each will require you to define a lambda expression taking some values as arguments and using them to calculate a specific result.

**Instructions 1/3**

**35 XP**

* Take xx and return x2x2 if x>0x>0 and 00, otherwise.

# Take x and return x squared if x > 0 and 0, otherwise

squared\_no\_negatives = lambda x: x\*\*2 if x > 0 else 0

print(squared\_no\_negatives(2.0))

print(squared\_no\_negatives(-1))

**Incorrect Submission**

Running the higlighted expression generated an error: name 'x' is not defined.

 Awesome, thanks for your feedback!

Take a list of integers nums and leave only even numbers.

# Take a list of integers nums and leave only even numbers

get\_even = lambda nums: [n for n in nums if not n % 2]

print(get\_even([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))

Take strings s1, s2 and list their common characters.

[**Take Hint (-9 XP)**](javascript:void(0))

# Take strings s1, s2 and list their common characters

common\_chars = lambda x, y: list(set(x).intersection(set(y)))

print(common\_chars('pasta', 'pizza'))

Traceback (most recent call last):

File "script.py", line 3, in <module>

print(squared\_no\_negatives(2.0))

TypeError: <lambda>() takes 0 positional arguments but 1 was given

<script.py> output:

4.0

0

<script.py> output:

[2, 4, 6, 8, 10]

<script.py> output:

['i', 'z', 's', 'a', 't', 'p']

<script.py> output:

['a', 'p']

In [1]:

**Incorrect Submission**

Did you keep only characters that appear in both strings?

+100 XP

Very good start on lambda expressions! As you can see, you can fold a pretty complicated code into a short statement and reuse it as much as you want.

**Converting functions to lambda expressions**

Convert these three normally defined functions into lambda expressions:

# Returns a bigger of the two numbers

def func1(x, y):

if x >= y:

return x

return y

# Returns a dictionary counting charaters in a string

def func2(s):

d = dict()

for c in set(s):

d[c] = s.count(c)

return d

# Returns a squared root of a sum of squared numbers

def func3(\*nums):

squared\_nums = [n\*\*2 for n in nums]

sum\_squared\_nums = sum(squared\_nums)

return math.sqrt(sum\_squared\_nums)

**Instructions 1/3**

**35 XP**

* Convert func1() to a lambda expression.

**Incorrect Submission**

Your code can not be executed due to a syntax error:  
invalid syntax (script.py, line 2).

# Convert func1() to a lambda expression

lambda1 = lambda x, y: x if x >= y else y

print(str(func1(5, 4)) + ', ' + str(lambda1(5, 4)))

print(str(func1(4, 5)) + ', ' + str(lambda1(4, 5)))

Convert func2() to a lambda expression.

[**Take Hint (-10 XP)**](javascript:void(0))

# Convert func2() to a lambda expression

lambda2 = lambda s: {c:s.count(c) for c in set(s)}

print(func2('DataCamp'))

print(lambda2('DataCamp'))

Convert func3() to a lambda expression.

[**Take Hint (-9 XP)**](javascript:void(0))

# Convert func3() to a lambda expression

lambda3 = lambda \*nums: math.sqrt(sum([n\*\*2 for n in nums]))

print(str(func3(3, 4)) + ', ' + str(lambda3(3, 4)))

print(str(func3(3, 4, 5)) + ', ' + str(lambda3(3, 4, 5)))

File "script.py", line 2

lambda1 = lambda x, y: x if x >= y

^

SyntaxError: invalid syntax

<script.py> output:

5, 5

5, 5

<script.py> output:

{'p': 1, 'D': 1, 'C': 1, 'm': 1, 'a': 3, 't': 1}

{'p': 1, 'D': 1, 'C': 1, 'm': 1, 'a': 3, 't': 1}

<script.py> output:

5.0, 5.0

7.0710678118654755, 7.0710678118654755

In [1]:

+100 XP

Good job! It is a very practical skill to understand when a normal function definition can be substituted with a lambda expression.

**Using a lambda expression as an argument**

Let's pass lambda expressions as arguments to functions. You will deal with the list .sort() method. By default, it sorts numbers in increasing order. Characters and strings are sorted alphabetically. The method can be defined as .sort(key=function). Here, key defines a mapping of each item in the considered list to a sortable object (*e.g.* a number or a character). Thus, the items in a list are sorted the way sortable objects are.

Your task is to define different ways to sort the list words using the key argument with a lambda expression.

**Instructions 1/3**

**35 XP**

* [1](javascript:void(0))

Sort words by string length.

[**Take Hint (-10 XP)**](javascript:void(0))

**Incorrect Submission**

Check your call of words.sort(). Did you correctly specify the argument key? The system wants to check the first lambda function but hasn't found it.

Did you find this feedback helpful?

[YesNo](javascript:void(0))

 Awesome, thanks for your feedback!

# Sort words by the string length

words.sort(key=lambda s: len(s))

print(words)

Sort words by the last character in a string.

# Sort words by the last character in a string

words.sort(key=lambda x: x[-1])

print(words)

Sort words by the total amount of characters a, b, and c (*e.g.*, the word 'cabana' has 3 a's, 1 b, and 1 c; in total, 5)

[**Take Hint (-9 XP)**](javascript:void(0))

# Sort words by the total amount of certain characters

words.sort(key=lambda x: x.replace('b','a').replace('c','a').count('a'))

print(words)

**Incorrect Submission**

Running the higlighted expression generated an error: 'list' object has no attribute 'replace'.

Did you sort the list by the total amount of characters 'a', 'b', and 'c'?

**Hint**

* Recall the string method .count().

# Sort words by the total amount of certain characters

words.sort(key=lambda s: s.count('a') + s.count('b') + s.count('c'))

print(words)

<script.py> output:

['car', 'bag', 'job', 'time', 'cell', 'call', 'area', 'item', 'unit', 'truck', 'phone', 'shape', 'plane', 'leader', 'height', 'tequila', 'chicken', 'country', 'service', 'creature', 'interview', 'advantage', 'government', 'atmosphere', 'transaction']

<script.py> output:

['car', 'bag', 'job', 'time', 'cell', 'call', 'area', 'item', 'unit', 'truck', 'phone', 'shape', 'plane', 'leader', 'height', 'tequila', 'chicken', 'country', 'service', 'creature', 'interview', 'advantage', 'government', 'atmosphere', 'transaction']

<script.py> output:

['tequila', 'area', 'job', 'time', 'service', 'phone', 'advantage', 'shape', 'atmosphere', 'creature', 'plane', 'bag', 'truck', 'cell', 'call', 'item', 'chicken', 'transaction', 'car', 'leader', 'government', 'height', 'unit', 'interview', 'country']

<script.py> output:

['interview', 'time', 'government', 'phone', 'item', 'height', 'unit', 'truck', 'tequila', 'cell', 'leader', 'country', 'service', 'job', 'shape', 'atmosphere', 'plane', 'car', 'chicken', 'bag', 'call', 'area', 'creature', 'transaction', 'advantage']

<script.py> output:

['time', 'item', 'unit', 'phone', 'height', 'interview', 'government', 'tequila', 'service', 'truck', 'job', 'shape', 'plane', 'leader', 'transaction', 'cell', 'country', 'atmosphere', 'area', 'chicken', 'creature', 'advantage', 'car', 'bag', 'call']

<script.py> output:

['interview', 'time', 'government', 'phone', 'item', 'height', 'unit', 'truck', 'tequila', 'cell', 'leader', 'country', 'service', 'job', 'shape', 'atmosphere', 'plane', 'car', 'chicken', 'bag', 'call', 'area', 'creature', 'transaction', 'advantage']

In [1]: words

Out[1]:

['car',

'truck',

'interview',

'tequila',

'time',

'cell',

'chicken',

'leader',

'government',

'transaction',

'country',

'bag',

'call',

'area',

'service',

'phone',

'advantage',

'job',

'shape',

'item',

'atmosphere',

'height',

'creature',

'plane',

'unit']

<script.py> output:

['interview', 'time', 'government', 'phone', 'item', 'height', 'unit', 'truck', 'tequila', 'cell', 'leader', 'country', 'service', 'job', 'shape', 'atmosphere', 'plane', 'car', 'chicken', 'bag', 'call', 'area', 'creature', 'transaction', 'advantage']

In [2]:

+70 XP

Great work! In the next lesson you will cover more functions that use lambda expressions in their arguments.

**Exercise**

**Exercise**

**The map() function**

Let's do some mapping!

Do you remember how zip() works? It merges given Iterables so that items with the same index fall into the same tuple. Moreover, the output is restricted by the shortest Iterable.

Your task is to define your own my\_zip() function with \*args depicting a variable number of Iterables, *e.g.* lists, strings, tuples *etc*. Rather than a zip object, my\_zip() should already return a list of tuples.

Comment: args should be checked whether they contain Iterables first. But we omit it for simplicity.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Retrieve Iterable lengths from args using map() and find the minimal length.
* Within the loop, create the mapping using map() to retrieve the elements in args with the same index i.
* Convert the mapping to a tuple and append it to the tuple\_list.

[**Take Hint (-30 XP)**](javascript:void(0))

**Hint**

* When defining lengths, args can be simply passed to the map() function as the second argument.
* Use a lambda expression in the definition of the map() function.

Did you find this hint helpful?

[YesNo](javascript:void(0))

def my\_zip(\*args):

# Retrieve Iterable lengths and find the minimal length

lengths = list(map(len, args))

min\_length = min(lengths)

tuple\_list = []

for i in range(0, min\_length):

# Map the elements in args with the same index i

mapping = map(lambda x: x[i], args)

# Convert the mapping and append it to tuple\_list

tuple\_list.append(tuple(mapping))

return tuple\_list

result = my\_zip([1, 2, 3], ['a', 'b', 'c', 'd'], 'DataCamp')

print(result)

+0 XP

Very good! Actually you could notice that sometimes map() can be substituted with a list comprehension. For example, list(map(lambda x: len(x), args)) can be re-written as [len(x) for x in args].

<script.py> output:

[(1, 'a', 'D'), (2, 'b', 'a'), (3, 'c', 't')]

In [1]:

**Exercise**

**Exercise**

**The filter() function**

Let's do some filtering! You will be given three corresponding tasks you have to complete. Use lambda expressions! And remember: the filter() function keeps all the elements that are mapped to the True value.

The variables string and spells are available in your workspace.

**Instructions 1/3**

**1 XP**

**Instructions 1/3**

**1 XP**

* Exclude all the numbers from nums divisible by 3 or 5.

**Incorrect Submission**

Check the first lambda function. To verify it, we reran it with the arguments (8). Did you filter out all numbers in nums divisible by 3 or 5?

Did you find this feedback helpful?

[YesNo](javascript:void(0))

**Hint**

* You might have to use the % operator.

# Exclude all the numbers from nums divisible by 3 or 5

print(nums)

fnums = filter(lambda x: x % 3 and x % 5, nums)

print(list(fnums))

solution.py

# Exclude all the numbers from nums divisible by 3 or 5

print(nums)

fnums = filter(lambda x: x % 3 != 0 and x % 5 != 0, nums)

print(list(fnums))

Return the string without its vowels ('y' is not a vowel in this case).

**Incorrect Submission**

We expected the following output: rdnry Lst Sqrs. Did you correctly create vowels and create fstring by returning string without vowels?

# Return the string without its vowels

print(string)

vowels = 'aiueo'

fstring = filter(lambda x: x.lower() not in vowels, string)

print(''.join(fstring))

Filter all the spells in spells with more than two 'a' characters.

# Filter all the spells in spells with more than two 'a's

print(spells)

fspells = filter(lambda x: x.count('a') > 2, spells)

print(list(fspells))

<script.py> output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

Traceback (most recent call last):

File "script.py", line 3, in <module>

fnums = filter(lambda x: not (x % 3 or x % 5))

TypeError: filter expected 2 arguments, got 1

<script.py> output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

[0, 15, 30, 45, 60, 75, 90]

<script.py> output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

[0, 3, 5, 6, 9, 10, 12, 15, 18, 20, 21, 24, 25, 27, 30, 33, 35, 36, 39, 40, 42, 45, 48, 50, 51, 54, 55, 57, 60, 63, 65, 66, 69, 70, 72, 75, 78, 80, 81, 84, 85, 87, 90, 93, 95, 96, 99, 100]

<script.py> output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

[0, 3, 5, 6, 9, 10, 12, 15, 18, 20, 21, 24, 25, 27, 30, 33, 35, 36, 39, 40, 42, 45, 48, 50, 51, 54, 55, 57, 60, 63, 65, 66, 69, 70, 72, 75, 78, 80, 81, 84, 85, 87, 90, 93, 95, 96, 99, 100]

<script.py> output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

[0, 3, 5, 6, 9, 10, 12, 15, 18, 20, 21, 24, 25, 27, 30, 33, 35, 36, 39, 40, 42, 45, 48, 50, 51, 54, 55, 57, 60, 63, 65, 66, 69, 70, 72, 75, 78, 80, 81, 84, 85, 87, 90, 93, 95, 96, 99, 100]

<script.py> output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

<script.py> output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

[1, 2, 4, 7, 8, 11, 13, 14, 16, 17, 19, 22, 23, 26, 28, 29, 31, 32, 34, 37, 38, 41, 43, 44, 46, 47, 49, 52, 53, 56, 58, 59, 61, 62, 64, 67, 68, 71, 73, 74, 76, 77, 79, 82, 83, 86, 88, 89, 91, 92, 94, 97, 98]

<script.py> output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]

[1, 2, 4, 7, 8, 11, 13, 14, 16, 17, 19, 22, 23, 26, 28, 29, 31, 32, 34, 37, 38, 41, 43, 44, 46, 47, 49, 52, 53, 56, 58, 59, 61, 62, 64, 67, 68, 71, 73, 74, 76, 77, 79, 82, 83, 86, 88, 89, 91, 92, 94, 97, 98]

In [1]: string

...

Out[1]: 'Ordinary Least Squares'

<script.py> output:

Ordinary Least Squares

Ordnry Lst Sqrs

<script.py> output:

Ordinary Least Squares

rdnry Lst Sqrs

In [2]: spells

Out[2]:

['riddikulus',

'obliviate',

'sectumsempra',

'avada kedavra',

'alohomora',

'lumos',

'expelliarmus',

'expecto patronum']

<script.py> output:

['riddikulus', 'obliviate', 'sectumsempra', 'avada kedavra', 'alohomora', 'lumos', 'expelliarmus', 'expecto patronum']

['avada kedavra']

In [3]:

66 XP

Great! No surprise that we got 'avada kedavra' as the only Harry Potter spell with more than two as!

**The reduce() function**

Now, it is time for some reduction! As before you'll be given three tasks to complete. Use lambda expressions!

The necessary functions from the functools module are already imported for you.

**Instructions 3/3**

**35 XP**

* Reverse a string using reduce().

# Reverse a string using reduce()

string = 'DataCamp'

inv\_string = string[::-1]

print('Inverted string = ' + inv\_string)

**Incorrect Submission**

Did you reverse the provided string?

* Find common items shared among all the sets in sets.

# Find common items shared among all the sets in sets

sets = [{1, 4, 8, 9}, {2, 4, 6, 9, 10, 8}, {9, 0, 1, 2, 4}]

common\_items = reduce(lambda x, y: x.intersection(y), sets)

print('common items = ' + str(common\_items))

* Convert a number sequence into a single number (*e.g.* [1, 2, 3] →→ 123).

# Convert a number sequence into a single number

nums = [5, 6, 0, 1]

num = str(nums)[1:-1].replace(', ', '') # reduce(\_\_\_\_)

print(str(nums) + ' is converted to ' + str(num))

Traceback (most recent call last):

File "script.py", line 3, in <module>

inv\_string = reduce(lambda x: x[::-1], string)

TypeError: <lambda>() takes 1 positional argument but 2 were given

<script.py> output:

Inverted string = pmaCataD

<script.py> output:

common items = {9, 4}

<script.py> output:

[5, 6, 0, 1] is converted to 5601

In [1]:

+100 XP

Nicely done! The functions map(), filter(), and reduce() are a serious weapon that you can use while solving a complex task. Now you are very well equiped!

**Exercise**

**Exercise**

**Calculate the number of function calls**

Let's consider a classic example of recursion – the Fibonacci sequence, represented by non-negative integers starting from 0 with each element F(n)F(n) equals the sum of the preceding two: 0, 1, 1, 2, 3, 5, 8, 13, 21, .... You are given a function that returns a tuple with the nn-th element of the sequence and the amount of calls to fib() used:

def fib(n):

if n < 2:

return (n, 1)

fib1 = fib(n-1)

fib2 = fib(n-2)

return (fib1[0] + fib2[0], fib1[1] + fib2[1] + 1)

How many calls to fib() are needed to calculate the 15th15th and 20th20th elements of the sequence?

**Instructions**

**50 XP**

**Possible Answers**

•

15th15th element: 3193 calls; 20th20th element: 35421 calls

•

15th15th element: 1973 calls; 20th20th element: 21891 calls

•

15th15th element: 3193 calls; 20th20th element: 21891 calls

•

15th15th element: 35421 calls; 20th20th element: 1973 calls

**Submit Answer**

[**Take Hint (-15 XP)**](javascript:void(0))

In [1]: fib

Out[1]: <function \_\_main\_\_.fib>

In [2]:

In [2]:

In [2]: In[2]

Out[2]: 'In[2]'

In [3]: In

Out[3]: ['', 'fib', 'In[2]', 'In']

In [4]: Out

Out[4]: {1: <function \_\_main\_\_.fib>, 2: 'In[2]', 3: ['', 'fib', 'In[2]', 'In', 'Out']}

In [5]: Out

Out[5]:

{1: <function \_\_main\_\_.fib>,

2: 'In[2]',

3: ['', 'fib', 'In[2]', 'In', 'Out', 'Out']}

In [6]: In, Out

Out[6]:

(['', 'fib', 'In[2]', 'In', 'Out', 'Out', 'In, Out'],

{1: <function \_\_main\_\_.fib>,

2: 'In[2]',

3: ['', 'fib', 'In[2]', 'In', 'Out', 'Out', 'In, Out']})

In [7]: type(In), type(Out)

Out[7]: (list, dict)

In [8]:

In [8]: type(In), type(Out)

Out[8]: (list, dict)

In [9]: exec(In[6])

In [10]: exec(In[6])

In [11]:

In [11]: In, Out

Out[11]:

(['',

'fib',

'In[2]',

'In',

'Out',

'Out',

'In, Out',

'type(In), type(Out)',

'type(In), type(Out)',

'exec(In[6])',

'exec(In[6])',

'In, Out'],

{1: <function \_\_main\_\_.fib>,

2: 'In[2]',

3: ['',

'fib',

'In[2]',

'In',

'Out',

'Out',

'In, Out',

'type(In), type(Out)',

'type(In), type(Out)',

'exec(In[6])',

'exec(In[6])',

'In, Out'],

6: (['',

'fib',

'In[2]',

'In',

'Out',

'Out',

'In, Out',

'type(In), type(Out)',

'type(In), type(Out)',

'exec(In[6])',

'exec(In[6])',

'In, Out'],

{...}),

7: (list, dict),

8: (list, dict)})

In [12]: fib(15), fib(20)

Out[12]: ((610, 1973), (6765, 21891))

In [13]:

+50 XP

Correct! Notice how big the difference is in function calls, even though they are only 5 indices away from each other. Therefore, recursion has to be used with caution. Too many calls can lead to memory errors.

**Exercise**

**Exercise**

**Calculate an average value**

We all know how to calculate an average value iteratively:

def average(nums):

result = 0

for num in nums:

result += num

return result/len(nums)

Could you provide a recursive solution? A formula for updating an average value given a new input might be handy:

x¯←xi+(n−1)x¯nx¯←xi+(n−1)x¯n

Here, x¯x¯ stands for an average value, xixi is a new supplied value which is used to update the average, and nn corresponds to the recursive call number (excluding the initial call to the function).

**Instructions**

**100 XP**

* Provide the base case for the algorithm.
* Define the recursive call for updating the average value.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Running the higlighted expression generated an error: name '\_\_\_\_' is not defined.

Check the definition of average(). Did you correctly specify the body? The system wants to check the first if statement but hasn't found it.

[**Show Answer (-70 XP)**](javascript:void(0))

**Hint**

* What is the average value of a single number?
* Did you follow the formula provided in the description?

# Calculate an average value of the sequence of numbers

def average(nums):

# Base case

if len(nums) == 1:

return nums[0]

# Recursive call

n = len(nums)

return (nums[0] + (n - 1) \* average(nums[1:])) / n

# Testing the function

print(average([1, 2, 3, 4, 5]))

+0 XP

Very good! Conversion of iterative algorithms to recursive ones and vice versa is a very frequent question in coding interviews.

<script.py> output:

3.0

In [1]:

**Exercise**

**Exercise**

**Approximate Pi with recursion**

The number ππ can be computed by the following formula:

π=4∑k=0∞(−1)k2k+1=4(11−13+15−17+19−…)π=4∑k=0∞(−1)k2k+1=4(11−13+15−17+19−…)

Your task is to write a recursive function to approximate ππ using the formula defined above (the approximation means that instead of infinity ∞∞, the sequence considers only a certain amount of elements nn).

Here are examples of ππ for some of the values of nn:  
n=0→π=4n=0→π=4  
n=1→π≈2.67n=1→π≈2.67  
n=2→π≈3.47n=2→π≈3.47

**Instructions 1/2**

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Write a lambda expression calculating the kk-th element in the series (without taking 4 into account).

[**Take Hint (-15 XP)**](javascript:void(0))

# Write an expression to get the k-th element of the series

get\_elmnt = lambda k: ((-1)\*\*k)/(2\*k+1)

* Specify the base case.
* Define the recursive call (n is the number of elements to consider).

[**Take Hint (-15 XP)**](javascript:void(0))

**Hint**

* The base case must certainly return a value.
* The recursive call must imply approaching the stopping criteria.
* Don't forget that eventually the elements are multiplied by 4.

# Write an expression to get the k-th element of the series

get\_elmnt = lambda k: ((-1)\*\*k)/(2\*k+1)

def calc\_pi(n):

curr\_elmnt = get\_elmnt(n)

# Define the base case

if n == curr\_elmnt:

return curr\_elmnt

# Make the recursive call

return calc\_pi(n)

# Compare the approximated Pi value to the theoretical one

print("approx = {}, theor = {}".format(calc\_pi(500), math.pi))

solution.py

# Write an expression to get the k-th element of the series

get\_elmnt = lambda k: ((-1)\*\*k)/(2\*k+1)

def calc\_pi(n):

curr\_elmnt = get\_elmnt(n)

# Define the base case

if n == 0:

return 4

# Make the recursive call

return 4 \* curr\_elmnt + calc\_pi(n-1)

# Compare the approximated Pi value to the theoretical one

print("approx = {}, theor = {}".format(calc\_pi(500), math.pi))

Traceback (most recent call last):

File "script.py", line 15, in <module>

print("approx = {}, theor = {}".format(calc\_pi(500), math.pi))

File "script.py", line 12, in calc\_pi

return calc\_pi(n)

File "script.py", line 12, in calc\_pi

return calc\_pi(n)

File "script.py", line 12, in calc\_pi

return calc\_pi(n)

[Previous line repeated 960 more times]

File "script.py", line 5, in calc\_pi

curr\_elmnt = get\_elmnt(n)

File "script.py", line 2, in <lambda>

get\_elmnt = lambda k: ((-1)\*\*k)/(2\*k+1)

File "script.py", line 96, in html\_plot\_trace

if plot\_class.is\_call\_match(func\_name, func\_filename, caller\_filename):

File "script.py", line 54, in is\_call\_match

func\_name == "open"

RecursionError: maximum recursion depth exceeded in comparison

In [1]:

<script.py> output:

approx = 3.143588659585789, theor = 3.141592653589793

In [1]:

+50 XP

Great job! Writing recursive functions for different mathematical series is a good exercise for understanding the concept of recursion. Keep practicing!

**Exercise**

**Exercise**

**Incorrect array initialization**

If you pass the following list [1, (2, 3), 4] to initialize a NumPy array, what would be the data type of the stored values?

The module numpy is already imported for you as np.

**Instructions**

**50 XP**

**Possible Answers**

int64

float64

<U1

object

**Submit Answer**

[**Take Hint (-15 XP)**](javascript:void(0))

+50 XP

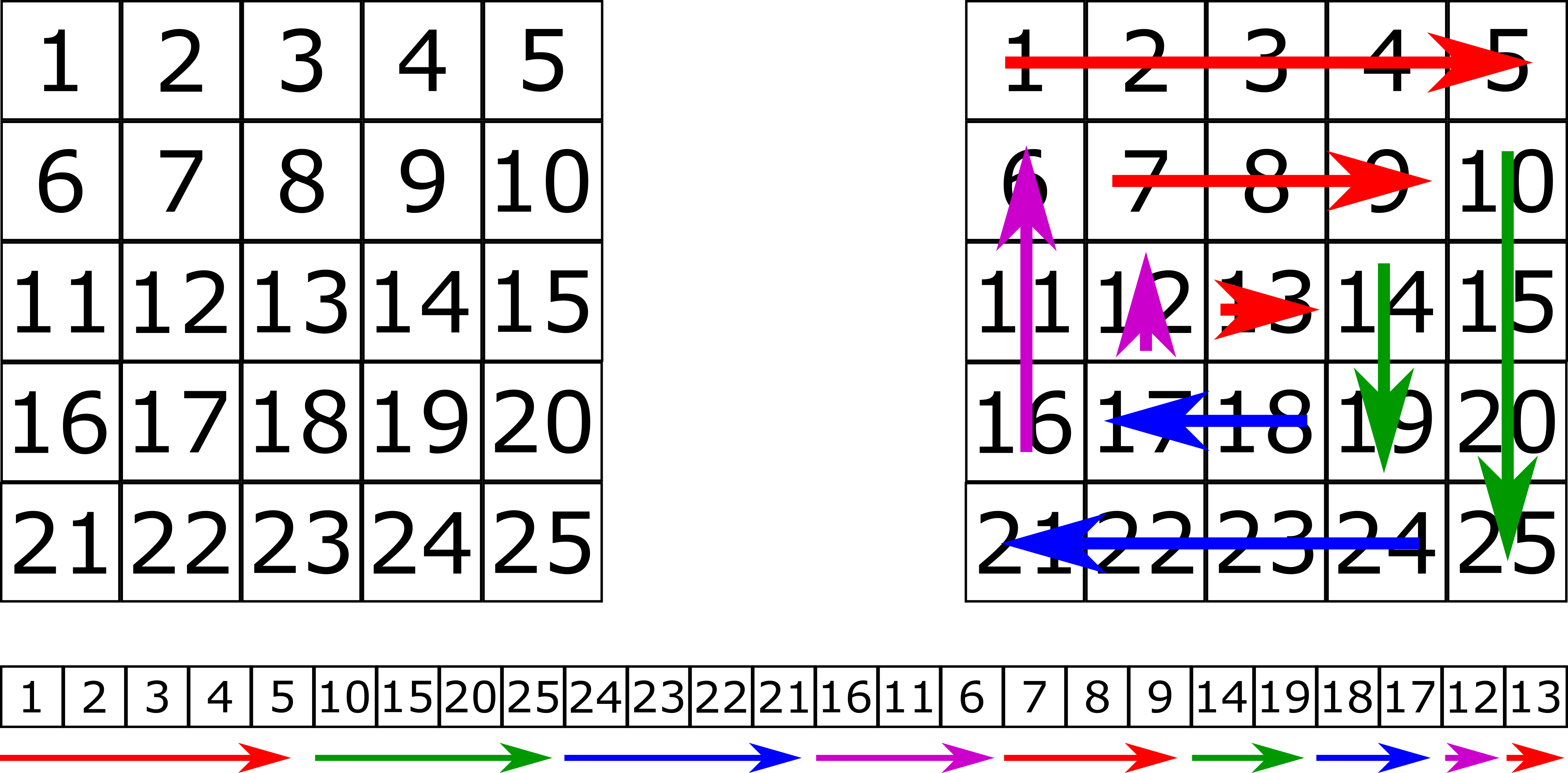
Correct. Since we have different data types in our list, all the elements are considered as an object.

**Exercise**

**Exercise**

**Accessing subarrays**

Let's access elements in NumPy arrays! Your task is to convert a square two-dimensional array square of size size to a list created by following a spiral pattern:



Rather than simply accessing certain slices, you will define a more general solution using a for loop (the solution should work for all the square two-dimensional arrays of odd size).

The module numpy is already imported as np.

You will need the reversed() function, which reverses an Iterable.

**Instructions**

**100 XP**

* Convert each part marked by a red arrow to a list.
* Convert each part marked by a green arrow to a list.
* Convert each part marked by a blue arrow to a list.
* Convert each part marked by a magenta arrow to a list.

[**Take Hint (-30 XP)**](javascript:void(0))

spiral = []

for i in range(0, size):

# Convert each part marked by any red arrow to a list

spiral += list(square[i, 0:size])

# Convert each part marked by any green arrow to a list

spiral += list(square[1:size, size-i-1])

# Convert each part marked by any blue arrow to a list

spiral += list(reversed(square[size-i-1, 0:size]))

# Convert each part marked by any magenta arrow to a list

spiral += list(reversed(square[1:size-i-1, 0]))

print(spiral)

solution.py

spiral = []

for i in range(0, size):

# Convert each part marked by a red arrow to a list

spiral += list(square[i, i:size-i])

# Convert each part marked by a green arrow to a list

spiral += list(square[i+1:size-i, size-i-1])

# Convert each part marked by a blue arrow to a list

spiral += list(reversed(square[size-i-1, i:size-i-1]))

# Convert each part marked by a magenta arrow to a list

spiral += list(reversed(square[i+1:size-i-1, i]))

print(spiral)

[[ 1 2 3 4 5]

[ 6 7 8 9 10]

[11 12 13 14 15]

[16 17 18 19 20]

[21 22 23 24 25]]

In [1]: spiral = []

for i in range(0, size):

# Convert each part marked by any red arrow to a list

spiral += list(square[i, 0:size])

# Convert each part marked by any green arrow to a list

spiral += list(square[1:size, size-i-1])

# Convert each part marked by any blue arrow to a list

spiral += list(reversed(square[size-i-1, 0:size]))

# Convert each part marked by any magenta arrow to a list

spiral += list(reversed(square[1:size-i-1, 0]))

print(spiral)

[1, 2, 3, 4, 5, 10, 15, 20, 25, 25, 24, 23, 22, 21, 16, 11, 6, 6, 7, 8, 9, 10, 9, 14, 19, 24, 20, 19, 18, 17, 16, 11, 6, 11, 12, 13, 14, 15, 8, 13, 18, 23, 15, 14, 13, 12, 11, 6, 16, 17, 18, 19, 20, 7, 12, 17, 22, 10, 9, 8, 7, 6, 21, 22, 23, 24, 25, 6, 11, 16, 21, 5, 4, 3, 2, 1]

<script.py> output:

[1, 2, 3, 4, 5, 10, 15, 20, 25, 24, 23, 22, 21, 16, 11, 6, 7, 8, 9, 14, 19, 18, 17, 12, 13]

In [2]:

+0 XP

Great job! You managed to unfold the matrix! You are very good in accessing array elements!