Comprehensions

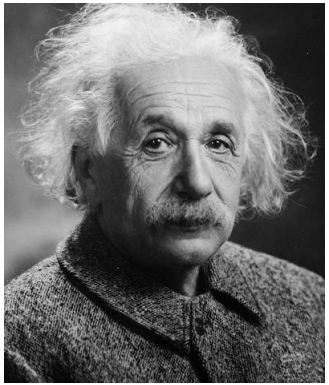
#7

Serdar ARITAN

Biomechanics Research Group,
Faculty of Sports Sciences, and
Department of Computer Graphics
Hacettepe University, Ankara, Turkey

“The only source of knowledge is experience.”

Albert Einstein



THE ROAD TO WISDOM

The road to wisdom?—Well, it's
plain and simple to express:
Err
and err
and err again,
but less
and less
and less.

***

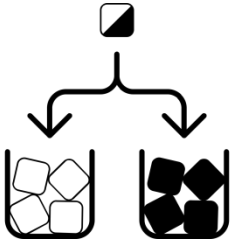
Piet Hein

Piet Hein was a Danish polymath, often writing under the Old Norse pseudonym Kumbel, meaning "**tombstone**". His short poems, known as gruks or grooks, first started to appear in the daily newspaper Politiken shortly after the German occupation of Denmark in April 1940 under the pseudonym "Kumbel Kumbell".

MS DOS

Measuring programming progress by lines of code is like measuring aircraft building progress by weight.

— Bill Gates



There are two ways of constructing a software design. One way is to make it **so simple** that there are **obviously no deficiencies**, and the other way is to make it **so complicated** that there are **no obvious deficiencies**.

— C.A.R. Hoare

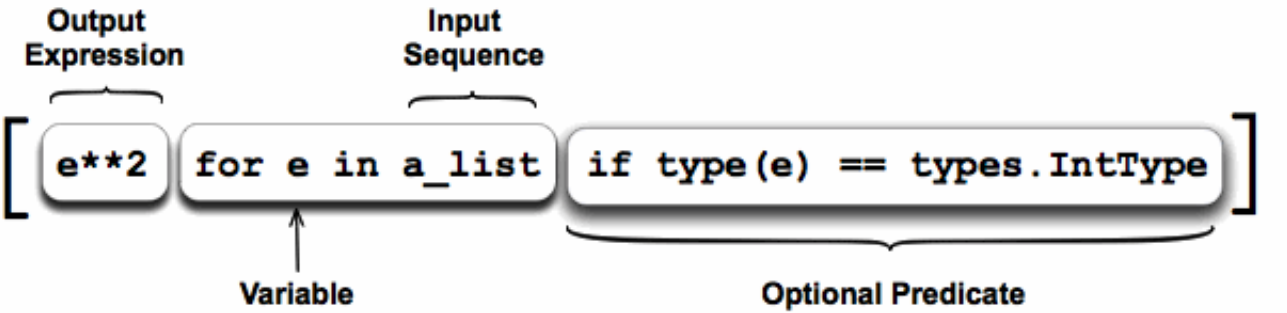
```
q = lambda l: q([x for x in l[1:] if x <= l[0]]) + [l[0]] + q([x for x in l if x > l[0]]) if l else []
```


I really enjoy learning
from my mistakes


"Well, now we know what not to do."

List Comprehensions

```
>>> a_list = [1, '4', 9, 'a', 0, 4]
>>> squared_ints = [e**2 for e in a_list if type(e) == types.IntType]
>>> print(squared_ints)
# [ 1, 81, 0, 16 ]
```



- The iterator part iterates through each member e of the input sequence a_list.
- The predicate checks if the member is an integer.
- If the member is an integer then it is passed to the output expression, squared, to become a member of the output list.

List Comprehensions

```
>>> x = [1, 2, 3, 4]
>>> x_squared = []
>>> for item in x:
...     x_squared.append(item * item)
>>> x_squared
[1, 4, 9, 16]
```

This sort of situation is so common that Python has a special shortcut for such operations, called a comprehension. The same thing as the previous code but is a list **comprehension**.

```
>>> x = [1, 2, 3, 4]
>>> x_squared = [item * item for item in x]
>>> x_squared
[1, 4, 9, 16]
```

List Comprehensions

```
[ expression for target in iterable ]

>>> [x ** 2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> res = []
>>> for x in [0, 1, 2]:
        for y in [100, 200, 300]:
                res.append(x + y)
>>> res
[100, 200, 300, 101, 201, 301, 102, 202, 302]
>>> res = [x + y for x in [0, 1, 2] for y in [100, 200, 300]]
>>> res
[100, 200, 300, 101, 201, 301, 102, 202, 302]
```

List Comprehensions

```
[ expression for target1 in iterable1 if condition1
              for target2 in iterable2 if condition2 ...
              for targetN in iterableN if conditionN ]
```

You can even use if statements to select items from the list:

```
>>> [x + y for x in 'spam' if x in 'sm' for y in 'SPAM' if y in
('P', 'A')]
['sP', 'sA', 'mP', 'mA']
>>> [x + y + z for x in 'spam' if x in 'sm'
                for y in 'SPAM' if y in ('P', 'A')
                for z in '123' if z > '1']
['sP2', 'sP3', 'sA2', 'sA3', 'mP2', 'mP3', 'mA2', 'mA3']
```

List Comprehensions

```
>>> x = [1, 2, 3, 4]
>>> x_squared = [item * item for item in x if item > 2]
>>> x_squared
[9, 16]
```

Dictionary comprehensions are similar, but you need to supply both a **key** and a **value**.

```
>>> x = [1, 2, 3, 4]
>>> x_squared_dict = {item: item * item for item in x}
>>> x_squared_dict
{1: 1, 2: 4, 3: 9, 4: 16}
```

List Comprehensions

```
sentence = 'BCO601 Python Programlama'
```

```
vowels = 'aeiou'
non_list = []
for l in sentence:
    if not l in vowels:
        non_list.append(l)
nonvowels = ''.join(non_list)
```

```
vowels = 'aeiou'
nonvowels = ''.join([L for L in sentence if not L in vowels])
'BCO601 Pythn Prgrmlm'
```

List Comprehensions

```
collection = list()
for datum in data_set:
    if condition(datum):
        collection.append(datum)
    else:
        new = modify(datum)
        collection.append(new)


collection = [d if condition(d) else modify(d) for d in data_set]
```

List Comprehensions

Write a program that calculates the sum of power of 3 the numbers between 1 to 9999.

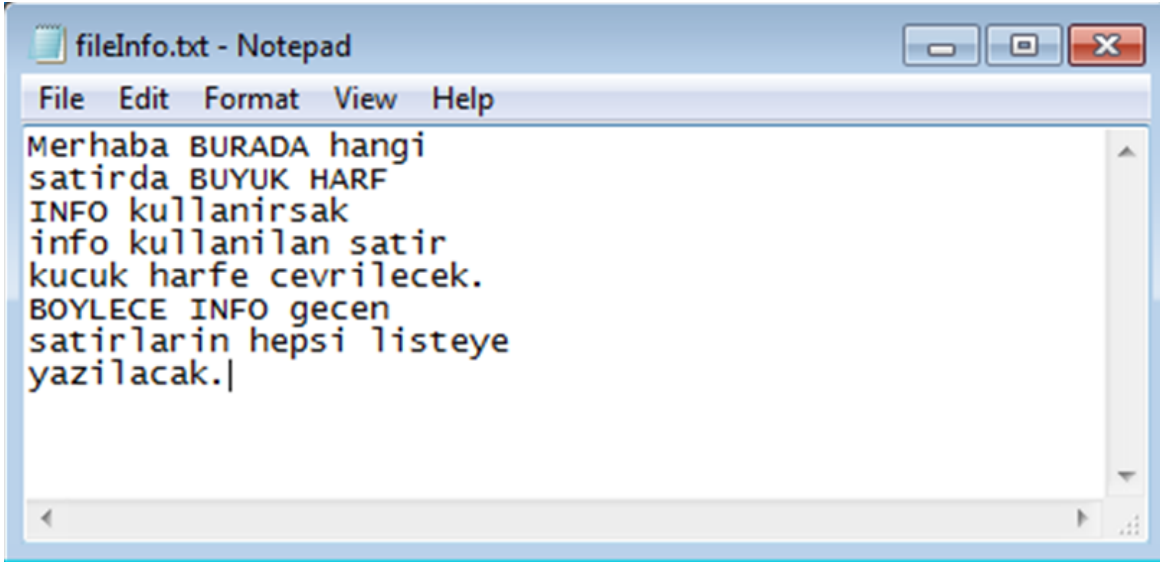
```
top = 0
for i in range(10000):
    top = top + i**3
print(top)
```

```
print(sum([x**3 for x in range(10000)])) # one line program
```

List Comprehensions

```
employees = {'Alice' : 100000,
             'Bob' : 99817,
             'Carol' : 122908,
             'Frank' : 88123,
             'Eve' : 93121}

top_earners = []

for key, val in employees.items():
    if val >= 100000:
        top_earners.append((key, val))
print(top_earners)
# [('Alice', 100000), ('Carol', 122908)]

top_earners_comp = [(k, v) for k, v in employees.items() if v >= 100000]

print(top_earners_comp)
```

List Comprehensions




```
>>> filename = 'fileInfo.txt'
>>> print([line.lower() for line in open(filename) if 'INFO' in line])

['info kullanirsak\n', 'boylece info gecen\n']
```

List Comprehensions

```
filename = "readFileDefault.py" # this code
f = open(filename)
lines = []
for line in f:
 lines.append(line.strip())
print(lines)
"""
['filename = "readFileDefault.py" # this code',
'',
'f = open(filename)',
'lines = []',
'for line in f:',
'lines.append(line.strip())',
'',
'print(lines)']
"""
print([line.strip() for line in open("readFileDefault.py")])
```

method `strip()` to remove any leading or trailing whitespace (otherwise, the newline character '\n' would appear in the strings).

List Comprehensions

```
text = '''
Call me Ishmael. Some years ago - never mind how long precisely - having
little or no money in my purse, and nothing particular to interest me
on shore, I thought I would sail about a little and see the watery part
of the world. It is a way I have of driving off the spleen, and regulating
the circulation. - Moby Dick'''
## One-Liner
w = [line for line in text.split() if len(line)>3]
print(w)

['Call', 'Ishmael.', 'Some', 'years', 'never', 'mind', 'long', 'precisely',
'having', 'little', 'money', 'purse,', 'nothing', 'particular', 'interest',
'shore,', 'thought', 'would', 'sail', 'about', 'little', 'watery', 'part',
'world.', 'have', 'driving', 'spleen,', 'regulating', 'circulation.', 'Moby',
'Dick']
```

List Comprehensions

```
text = '''
Call me Ishmael. Some years ago - never mind how long precisely - having
little or no money in my purse, and nothing particular to interest me
on shore, I thought I would sail about a little and see the watery part
of the world. It is a way I have of driving off the spleen, and regulating
the circulation. - Moby Dick'''
## One-Liner
w = [[x for x in line.split() if len(x)>3] for line in text.split('\n')]
print(w)
```

```
[[], ['Call', 'Ishmael.', 'Some', 'years', 'never', 'mind', 'long',
'precisely', 'having'], ['little', 'money', 'purse,', 'nothing', 'particular',
'interest'], ['shore,', 'thought', 'would', 'sail', 'about', 'little',
'watery', 'part'], ['world.', 'have', 'driving', 'spleen,', 'regulating'],
['circulation.', 'Moby', 'Dick']]
```

List Comprehensions

```
# Transform string into HTML.
def html(s):
    return "<b>" + s.capitalize() + "</b>"

# Input string.
input = ["rabbit", "mouse", "gorilla", "giraffe"]

# List comprehension.
list = [html(x) for x in input]

# Result list.
print(list)
```

```
['<b>Rabbit</b>', '<b>Mouse</b>', '<b>Gorilla</b>',
'<b>Giraffe</b>']
```

List Comprehensions

```
## Data (daily stock prices ($))
price = [[9.9, 9.8, 9.8, 9.4, 9.5, 9.7],
         [9.5, 9.4, 9.4, 9.3, 9.2, 9.1],
         [8.4, 7.9, 7.9, 8.1, 8.0, 8.0],
         [7.1, 5.9, 4.8, 4.8, 4.7, 3.9]]
## One-Liner
sample = [line[::2] for line in price]
## Result
print(sample)
```

```
[[9.9, 9.8, 9.5],
 [9.5, 9.4, 9.2],
 [8.4, 7.9, 8.0],
 [7.1, 4.8, 4.7]]
```

List Comprehensions

Our goal is to replace every other string with the string immediately in front of it;

```
## Data
visitors = ['Firefox', 'corrupted', 'Win10', 'corrupted',
            'Safari', 'corrupted', 'MacOs', 'corrupted',
            'Chrome', 'corrupted', 'Linux', 'corrupted']
## One-Liner
visitors[1::2] = visitors[::2]
## Result
print(visitors)

['Firefox', 'Firefox', 'Win10', 'Win10', 'Safari', 'Safari',
'MacOs', 'MacOs', 'Chrome', 'Chrome', 'Linux', 'Linux']
```

List Comprehensions

```
## Data
column_names = ['name', 'salary', 'job']
db_rows = [('Alice', 180000, 'data scientist'),
           ('Bob', 99000, 'mid-level manager'),
           ('Frank', 87000, 'CEO')]
## One-Liner
db = [dict(zip(column_names, row)) for row in db_rows]
## Result
print(db)
```

```
[{'name': 'Alice', 'salary': 180000, 'job': 'data scientist'},
 {'name': 'Bob', 'salary': 99000, 'job': 'mid-level manager'},
 {'name': 'Frank', 'salary': 87000, 'job': 'CEO'}]
```

Dictionary Comprehensions

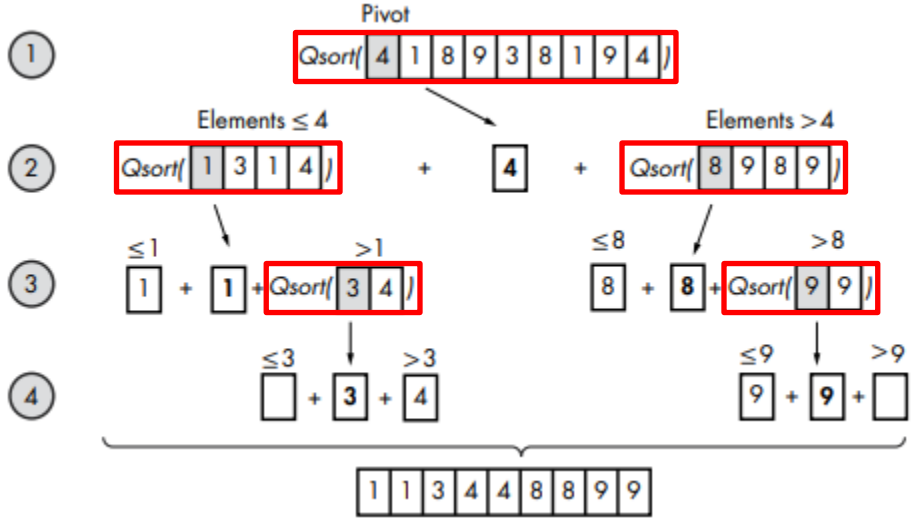
```
d = {'sea':'blue', 'grass':'green','fire':'red'}

for k in d:
    print(k)


d = {k : d[k] for k in d if not k.startswith('f')}

print(d)

{'sea': 'blue', 'grass': 'green'}
```

Dictionary Comprehensions

```
>>> import os

>>> ext = {os.path.splitext(filename)[1] for filename in os.listdir('.')}

>>> ext
{'', '.exe', '.txt', '.dll'}

>>> dir = {filename: os.path.getsize(filename) for filename in os.listdir('.')}

>>> {'DLLs': 8192, 'Doc': 0, 'include': 28672, 'Lib': 49152, 'libs': 0,
'LICENSE.txt': 36874, 'NEWS.txt': 1745082, 'python.exe': 103704, 'python3.dll':
68376, 'python312.dll': 6927640, 'pythonw.exe': 102168, 'Scripts': 4096, 'tcl':
4096, 'vcruntime140.dll': 119192, 'vcruntime140_1.dll': 49528}
```

Recursive Quicksort Algorithm

`Quicksort` sorts a list by recursively dividing the big problem into smaller problems and combining the solutions from the smaller problems in a way that it solves the big problem. To solve each smaller problem, the same strategy is used recursively: the smaller problems are divided into even smaller subproblems, solved separately, and combined, placing **`Quicksort`** in the class of Divide and Conquer algorithms.



Recursive Quicksort Algorithm

```
## The Data
unsorted = [33, 2, 3, 45, 6, 54, 33, 54, 13, 1, 7, 67]
## The One-Liner
q = lambda L: q([x for x in L[1:] if x <= L[0]]) + [L[0]] + q([x for x in L if x > L[0]]) if L else []
## The Result
print(q(unsorted))
```

```
[1, 2, 3, 6, 7, 13, 33, 33, 45, 54, 54, 67]
```

Dictionary Examples

1. Write a program that counts the letter in the text.

2. Write a function that converts the Turkish letters to English equivalents.

```
>>> eqtable={
        'ş':'s', 'Ş':'S',
        'ğ':'g', 'Ğ':'G',
        'ı' : 'i', 'İ':'I',
        'ü':'u', 'Ü':'U',
        'ö':'o', 'Ö':'O'
        }
```

Map, Lambda, and Filter

The `map()` function applies a function to every member of an iterable and returns the result. If we simply call a function inside a for loop, the built-in higher order function `map()` comes to our aid:

```
for e in it: # statement-based loop
    func(e)
```

The following code is entirely equivalent to the functional version

```
map(func, it) # map()-based "loop"
```

Map, Lambda, and Filter

The `map()` function applies a function to every member of an iterable and returns the result. Typically, one would use an anonymous inline function as defined by lambda, but it is possible to use any function

```
>>> def square(x):
...     return x**2
...

>>> squares = map(square, range(10))
>>> print (*squares)
0 1 4 9 16 25 36 49 64 81
>>> squares = map(lambda x: x**2, range(10))
>>> print (*squares)
0 1 4 9 16 25 36 49 64 81
```

Map, Lambda, and Filter

```
## Data
txt = ['lambda functions are anonymous functions.',
 'anonymous functions dont have a name.',
 'functions are objects in Python.']

## One-Liner
mark = map(lambda s: (True,s) if 'anonymous' in s else (False,s), txt)
## Result
print(list(mark))
```

```
[(True, 'lambda functions are anonymous functions.'), (True,
'anonymous functions dont have a name.'), (False, 'functions are
objects in Python.')]
```

Map, Lambda, and Filter

```
>>> def fahrenheit(T):
...     return ((float(9)/5)*T + 32)
...
>>> def celsius(T):
...     return (float(5)/9)*(T-32)
...
>>> temperatures = (36.5, 37, 37.5, 38, 39)
>>> F = map(fahrenheit, temperatures)
>>> C = map(celsius, F)
>>> temp_in_F = list(map(fahrenheit, temperatures))
>>> temp_in_C = list(map(celsius, temp_in_F))
>>> print(temp_in_F)
[97.7, 98.60000000000001, 99.5, 100.4, 102.2]
>>> print(temp_in_C)
[36.5, 37.00000000000001, 37.5, 38.00000000000001, 39.0]
>>>
```

Map, Lambda, and Filter

```
>>> C = [39.2, 36.5, 37.3, 38, 37.8]
>>> F = list(map(lambda x: (float(9)/5)*x + 32, C))
>>> print(F)
[102.56, 97.7, 99.14, 100.4, 100.03999999999999]
>>> C = list(map(lambda x: (float(5)/9)*(x-32), F))
>>> print(C)
[39.2, 36.5,37.300000000000004, 38.00000000000001, 37.8]
>>>
```

Map, Lambda, and Filter

`map()` can be applied to more than one list. The lists have to have the same length. `map()` will apply its lambda function to the elements of the argument lists, i.e. it first applies to the elements with the 0th index, then to the elements with the 1st index until the nth index is reached

```
>>> a = [1,2,3,4]
>>> b = [17,12,11,10]
>>> c = [-1,-4,5,9]
>>> list(map(lambda x,y:x+y, a,b))
[18, 14, 14, 14]
>>> list(map(lambda x,y,z:x+y+z, a,b,c))
[17, 10, 19, 23]
>>> list(map(lambda x,y,z : 2.5*x + 2*y - z, a,b,c))
[37.5, 33.0, 24.5, 21.0]
>>>
```

Map, Lambda, and Filter

The function `filter(function, sequence)` offers an elegant way to filter out all the elements of a sequence "sequence", for which the function returns True.

```
>>> f = [0,1,1,2,3,5,8,13,21,34,55]
>>> odd_numbers = list(filter(lambda x: x % 2, f))
>>> print(odd_numbers)
[1, 1, 3, 5, 13, 21, 55]
>>> even_numbers = list(filter(lambda x: x % 2 == 0, f))
>>> print(even_numbers)
[0, 2, 8, 34]
>>> even_numbers = list(filter(lambda x: not (x % 2), f))
>>> print(even_numbers)
[0, 2, 8, 34]
>>> even_numbers = list(filter(lambda x: x % 2 - 1, f))
>>> print(even_numbers)
[0, 2, 8, 34]
```

```
#!/usr/bin/python3
def main():
    print("This is main function")
    yumyum()

def yumyum():
    print("yumyum")

if __name__ == "__main__": main()
```

```
>> This is main function
   yumyum
```

if __name__ == "__main__": main()

```
print("This is main function")
yumyum()

def yumyum():
    print("yumyum")

if __name__ == "__main__": main()

>> This is main function
   Traceback (most recent call last):
  File "D:/Lectures/BCO 601 Python Programming/Loops and
Function/fonksiyon3.py", line 4, in <module>
    yumyum()
NameError: name 'yumyum' is not defined
```

if __name__ == "__main__": main()

```
#!/usr/bin/python3
def main():
        s = 'this is main string'
        for c in s:
                if c == 's': continue
                print(c, end = ' ')

if __name__ == "__main__": main()
```

```
>> t h i   i   m a i n   t r i n g
```

```
>>> soz = "bak göreceksin Python öğreneceğim"
>>> print(soz[::-1])
miğecenerğö nohtyP niskecerög kab
>>> reversed(soz)
<reversed object at 0x000000000341FD30>
>>> print(reversed(soz))
<reversed object at 0x000000000341FE10>
>>> for i in reversed(soz):
        print(i)
m
i
ğ
e
.
>>> print(*reversed(soz))
m i ğ e c e n e r ğ ö   n o h t y P   n i s k e c e r ö g   k a b
```

*

```
>>> print(*”Hacettepe")
H a c e t t e p e
>>> print(*"TBMM", sep=".")
T.B.M.M
>>> print(*"abcçdefggh", sep="/")
a/b/c/ç/d/e/f/g/g/h
>>> range(10)
range(0, 10)
>>> print(range(10))
range(0, 10)
>>> print(*range(10))
0 1 2 3 4 5 6 7 8 9
>>> print(*range(10), sep=", ")
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

*

```
>>> a, b, c = "X...Y"
Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    a, b, c = "X...Y"
ValueError: too many values to unpack (expected 3)
>>> a, *b, c = "X...Y"
>>> b
['.', '.', '.']
```

enumerate()

```
flavor_list = ['vanilla', 'chocolate', 'pecan', 'strawberry']

for i in range(len(flavor_list)):
    print(f'{i + 1}: {flavor_list[i]}')

for i, flavor in enumerate(flavor_list):
    print(f'{i + 1}: {flavor}')

for i, flavor in enumerate(flavor_list, 1):
    print(f'{i}: {flavor}')
```

```
1: vanilla
2: chocolate
3: pecan
4: strawberry
```

enumerate()

```
>>> enumerate('serdar')
<enumerate object at 0x0000000003453798>
>>> print(enumerate('serdar'))
<enumerate object at 0x0000000003453750>
>>> print(*enumerate('serdar'))
(0, 's') (1, 'e') (2, 'r') (3, 'd') (4, 'a') (5, 'r')
>>> for sira, harf in enumerate('serdar'):
        print(sira, harf)

0 s
1 e
2 r
3 d
4 a
5 r
>>> for sira, harf in enumerate('serdar',1):
        print(sira, harf)

1 s
```

HomeWork

On some basic cell phones, text messages can be sent using the numeric keypad. Because each key has multiple letters associated with it, multiple key presses are needed for most letters. Pressing the number once generates the first character listed for that key. Pressing the number 2, 3, 4 or 5 times generates the second, third, fourth or fifth character.

Key	Symbols
1	. , ? ! :
2	A B C
3	D E F
4	G H I
5	J K L
6	M N O
7	P Q R S
8	T U V
9	W X Y Z
0	*space*

Write a program that displays the key presses needed for a message entered by the user. Construct a dictionary that maps from each letter or symbol to the key presses needed to generate it. Then use the dictionary to create and display the presses needed for the user's message. For example, if the user enters `Hello, World!` then your program should output `4433555555666110966677755531111`.

HomeWork

Amateur radio, also known as ham radio, is the use of radio frequency spectrum for purposes of non-commercial exchange of messages, wireless experimentation, self-training, private recreation, radiosport, contesting, and emergency communication.

HomeWork

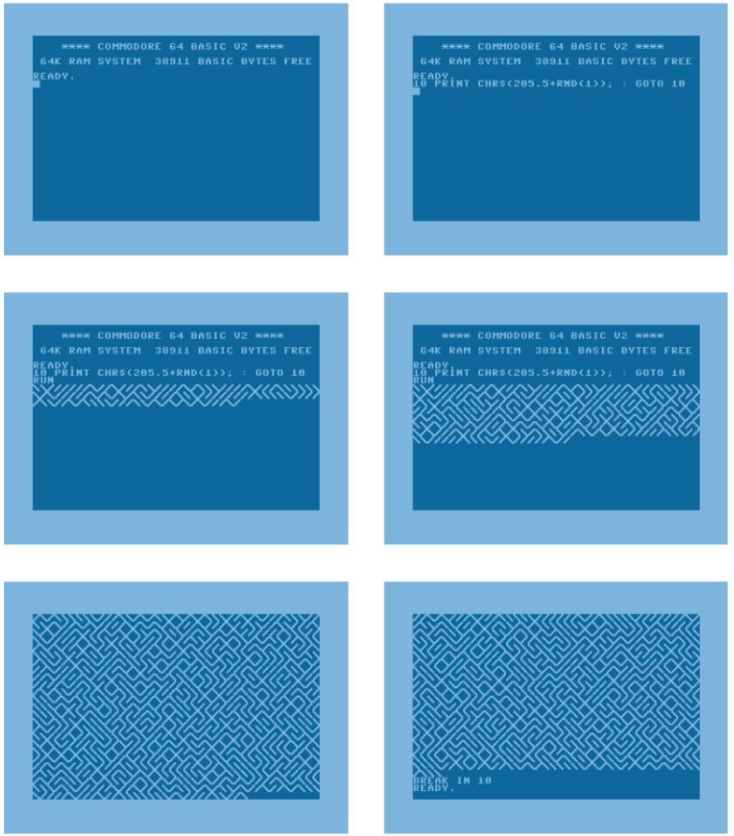
A spelling alphabet is a set of words, each of which stands for one of the 26 letters in the alphabet. While many letters are easily misheard over a low quality or noisy communication channel, the words used to represent the letters in a spelling alphabet are generally chosen so that each sounds distinct and is difficult to confuse with any other. The NATO phonetic alphabet is a widely used spelling alphabet.

Write a program that reads a word from the user and then displays its phonetic spelling. For example, if the user enters `Hello` then the program should output `Hotel Echo Lima Lima Oscar`.

Letter	Word	Letter	Word	Letter	Word
A	Alpha	J	Juliet	S	Sierra
B	Bravo	K	Kilo	T	Tango
C	Charlie	L	Lima	U	Uniform
D	Delta	M	Mike	V	Victor
E	Echo	N	November	W	Whiskey
F	Foxtrot	O	Oscar	X	Xray
G	Golf	P	Papa	Y	Yankee
H	Hotel	Q	Quebec	Z	Zulu
I	India	R	Romeo		

“10 PRINT CHR$(205.5+RND(1)); : GOTO 10” (MIT Press, 2012)

This book believe that analyzing one line of code is valuable because it gives us an understanding of the cultural, social and political knowledge about code and how we use code.



https://www.youtube.com/watch?v=m9joBLOZVEo