# Week 5

Dictionaries

# Dictionaries

# Quick Dictionary Exercise

**Code**

```
a = (("A",2), ("B",3), (1,4))
dict_a = dict(a)
print(dict_a)
print(dict_a[2])

b = [[1,"A"], [(2,3),4]]
dict_b = dict(b)
print(dict_b)
print(dict_b[(2,3)])
```

**Output**

```
{'A': 2, 'B': 3, 1: 4}
KeyError

{1: 'A', (2, 3): 4}
4
```

```
dict_b = {1: ‘A’, (2, 3): 4}
```

# Quick Dictionary Exercise

**Code**

```
for key in dict_b.keys():
    print(key)

for val in dict_b.values():
    print(val)

for k,v in dict_b.items():
    print(k, v)
```

**Output**

```
1
(2, 3)

A
4

1 A
(2, 3) 4
```

# Quick Dictionary Exercise

| **Code** | **Output** |
| --- | --- |
| `del dict_b[(2, 3)]` | |
| `print(dict_b)` | `{1: 'A'}` |
| `del dict_b[2]` | `KeyError` |
| `print(dict_b)` | `{1: 'A'}` |

# Quick Dictionary Exercise

| **Code** | **Output** |
| --- | --- |
| `print(tuple(dict_a.keys()))` | `('A', 'B', 1)` |
| `print(list(dict_a.values()))` | `[2, 3, 4]` |

# Quick Dictionary Exercise

**Code**

```
dict_c = {1: {2: 3}, 4: 5}
dict_d = dict_c.copy()


dict_d[4] = 9
dict_d[1][2] = 9
print(dict_d)
print(dict_c)
```

**Output**

```
{1: {2: 9}, 4: 9}
{1: {2: 9}, 4: 5}
```

```
dict_c = {1: {2: 9}, 4: 5}
```

# Quick Dictionary Exercise

**Code**

```
del dict_c
print(dict_c)
```

**Output**

```
NameError
```

# Dictionary

- Definition
  - *Mutable* set of key: value pairs
  - Enclosed in braces
  - Objects are separated by commas
- Creation
  - Empty dictionary `{}`
  - Initialized dictionary `{key1: elem1, key2: elem2, ... }`
  - From iterable e.g. tuple/list `dict(iterable_of_pairs)`
    - Every element in the iterable must be a pair
    - The first element in the pair will be the key
    - The second element in the pair will be the value

# Dictionary

- Access
  - `x[key]` throws an error if `key` is not in `x`
  - `x.get(key, default=None)` returns `default` if `key` is not in `x`
- Insertion/Modification
  - `x[key] = value`
- Deletion
  - `del x[key]` deletes the entry corresponding to `key`
  - `x.clear()` removes all entries in `x`
  - `del x` deletes `x`

# Dictionary

- (Other) Operations
  - `len(x)` returns the number of keys in `x`
  - `key in x` returns `True` if `key` is in `x`, and `False` otherwise
  - `for key in x` iterates over the keys of `x`; each element is stored in `key`
  - `max(x)` returns the maximum key in `x`
  - `min(x)` returns the minimum key in `x`

  - `x.keys()` returns a view of the keys of `x`
  - `x.values()` returns a view of the values of `x`
  - `x.items()` returns a view of the key:value pairs of `x`

# Exercise: Anagram

# Problem

- Anagram: word or phrase formed by rearranging the letters of an original word or phrase, typically using all the letters of the original word or phrase exactly once
- Examples:
  - “nag a ram” is an anagram for “anagram”
  - “eleven plus two” is an anagram for “twelve plus one”
- Write a function `is_anagram(word1, word2)` that returns `True` if the two words are anagrams of each other, otherwise return `False`

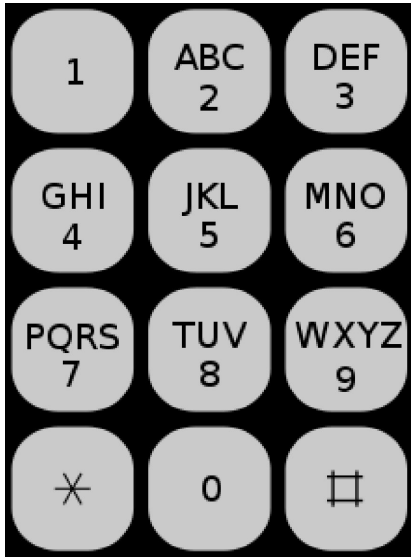
By using Dictionary

# Exercise: T9

# Problem

- Old mobile phones have numerical keypads where every character is associated with exactly one number, e.g.
  - “a” is associated with 2
  - “ ” is associated with 0

- We can represent the keypad in two different ways:
  1. A ***list*** where each element is a string of characters that are all associated with the number which is the element’s index
     `keyL = [“ ”, “”, “abc”, “def”, “ghi”, “jkl”, “mno”, “pqrs”, “tuv”, “wxyz”]`
  2. A ***dictionary*** where the keys are the characters and the values are their associated numbers
     keyD = `{“a”: 2, “b”: 2, ..., “z”: 9, “ ”: 0}`

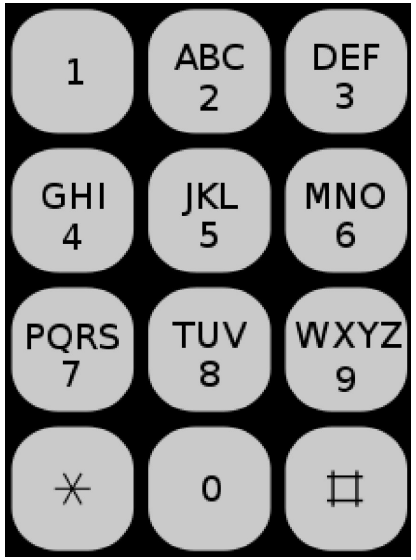


# Tasks

Suppose there are other keypads which can support any printable character in the ASCII table, associated with any number that is a nonnegative integer.

1. Write a function `to_dict(keyL)` which
   - takes in some keypad as a list, and
   - returns it as a dictionary

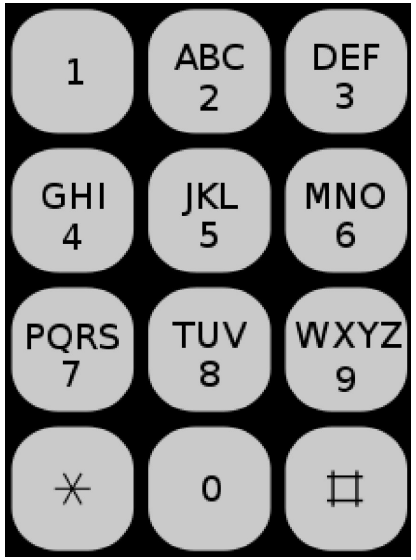
2. Write a function `to_list(keyD)` which
   - takes in some keypad as a dictionary, and
   - returns it as a list



# Problem

- Since typing text was tedious with old mobile phones, the T9 system was created as a predictive text technology.

- The system works as follows:
  - Every phrase is composed of letters
  - Every letter can be mapped to a number
  - Given a sequence of keypresses (numbers), we can predict the desired phrase

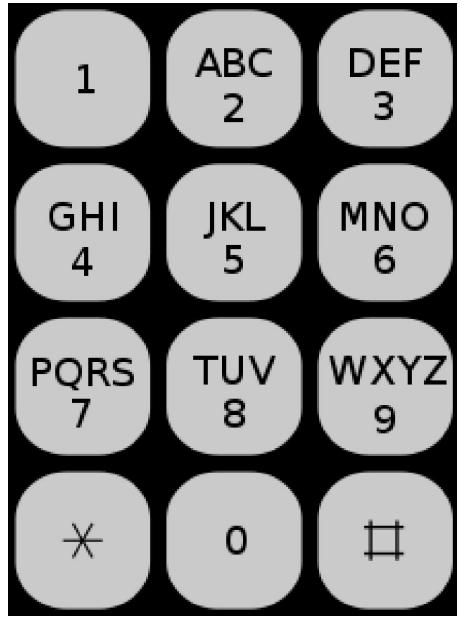


# Tasks

You may assume that both `keyL` and `keyD` are already initialized.

1. Write a function `to_nums(word)` that
   - takes in an input string, and
   - returns an integer representing the sequence of keypresses (numbers) that can be predicted as the desired phrase

   E.g. `to_nums("i luv u")` returns `4058808`




2. Write a function `to_letters(num)` that
   - takes in a nonnegative number, and
   - returns a list of all possible phrases that can be predicted (in any order)