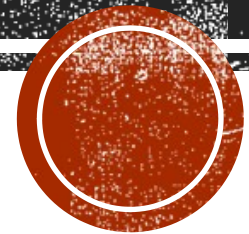


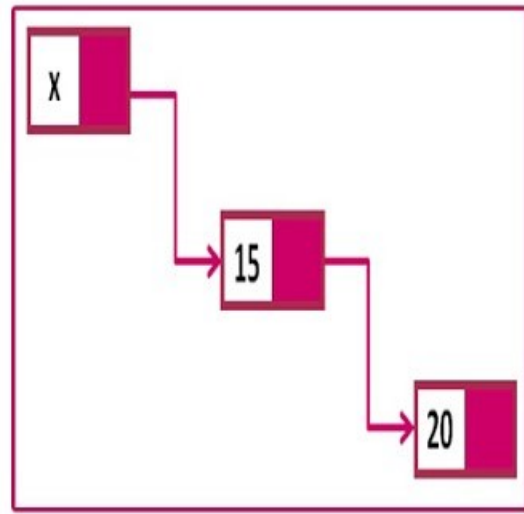
# DATA STRUCTURES



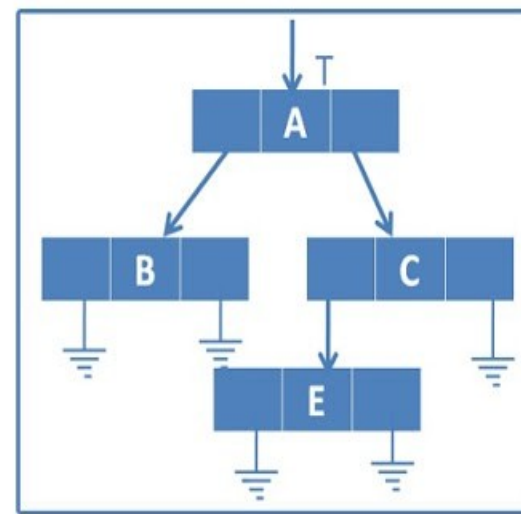
# REVIEW: WHAT ARE DATA STRUCTURES?

In computer science, a **data structure** is a particular way of **organizing data** in a computer so that it can be used efficiently.

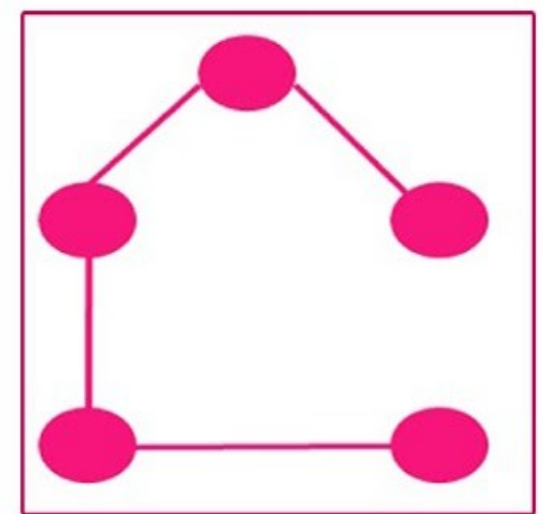




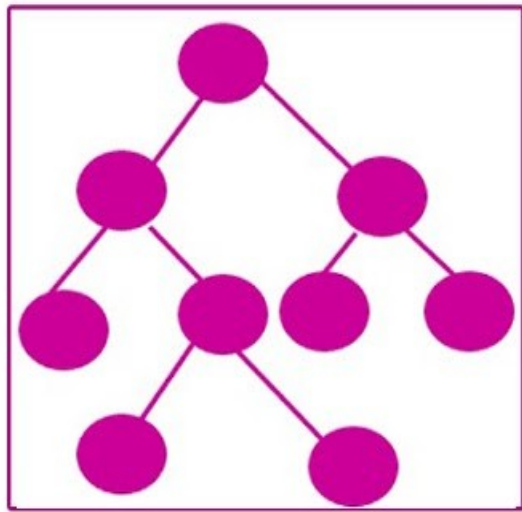
Link list



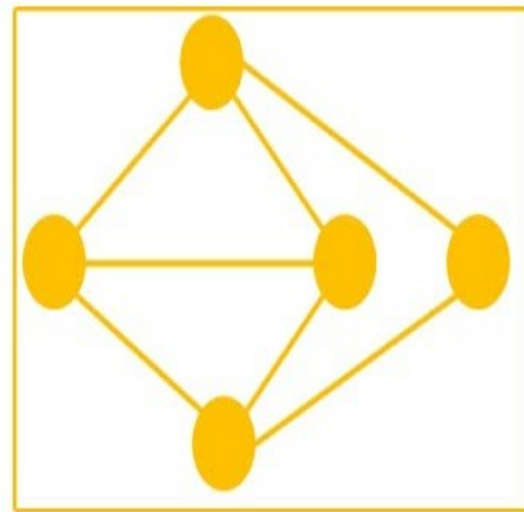
list



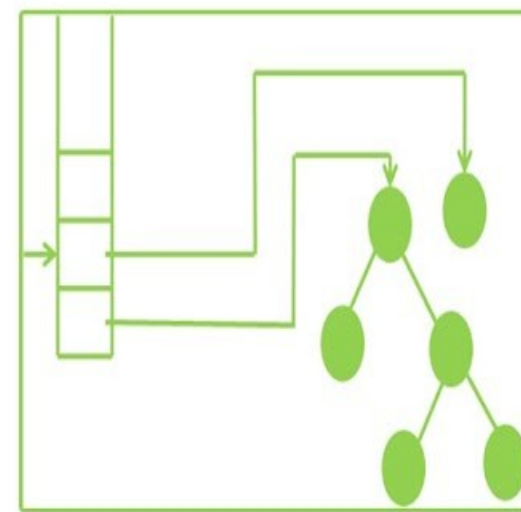
spanning tree



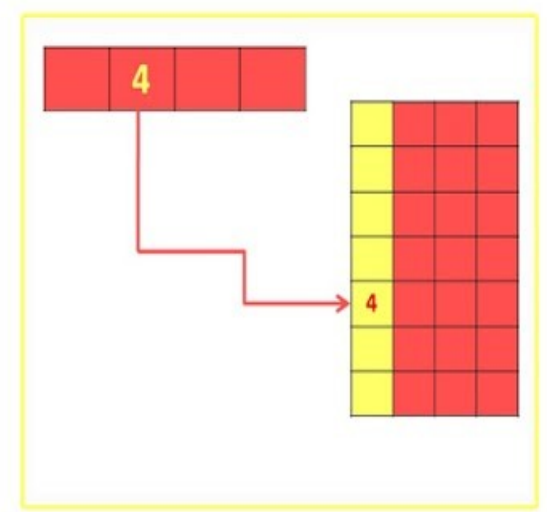
Tree



Graph



Stack



Hashing



# REVIEW: LISTS

- A List is a kind of Collection
- A collection allows us to put many values in a single “variable”
- A collection is nice because we can carry many values around in one convenient package.



# REVIEW: LIST OPERATIONS

```
>>> a_list = ["a", "b", "c", "d", "e", "f"]
```

```
>>> a_list[1:3]
```

```
['b', 'c']
```

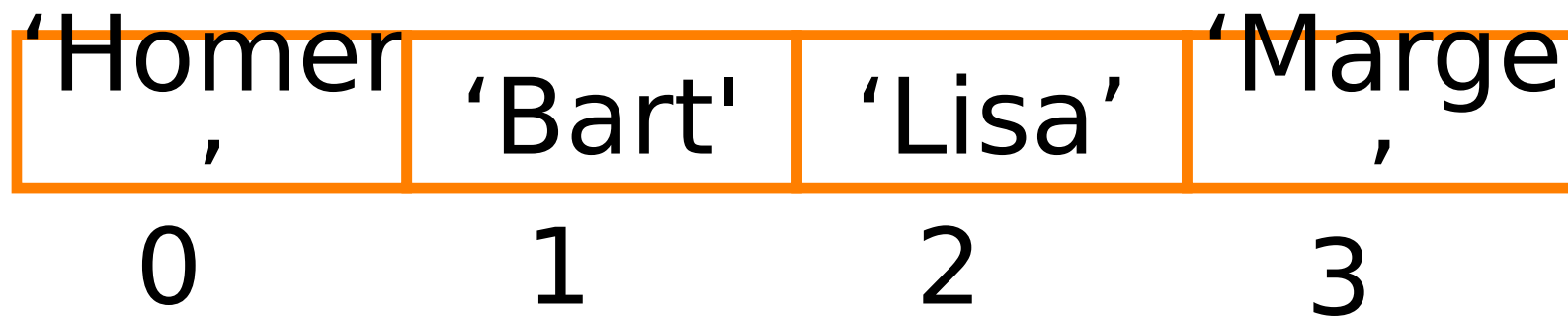
```
>>> a_list[:4]
```

```
['a', 'b', 'c', 'd']
```



# REVIEW: LOOKING INSIDE LISTS

Just like strings, we can get at any single element in a list using an index specified in **square brackets**



# LIST ARGUMENTS

Write a function that determines the length of a list



# LIST APPEND METHOD

```
>>> t1 = [1, 2]
```

```
>>> t1.append(3)
```

```
>>> print(t1)
```

```
[1, 2, 3]
```



# LIST INSERT METHOD

```
>>> l = [1, 2]
```

```
>>> l.insert(2,4)
```

```
>>> print(l)
```

```
[1, 2, 4]
```



# LIST INSERT METHOD

```
>>> l = [1, 2]
```

```
>>> l.insert(23,4)
```

```
>>> print(l)
```

```
[1, 2, 4]
```



# LIST INSERT METHOD

```
>>> l = [1, 2]
```

```
>>> l.insert(1,4)
```

```
>>> print(l)
```

```
[1, 4, 2]
```



# LIST METHODS

```
>>> t1 = [1, 2]
```

```
>>> t2 = t1.append(3)
```

```
>>> print(t1)
```

```
[1,2,3]
```

```
>>> print(t2)
```

```
None
```

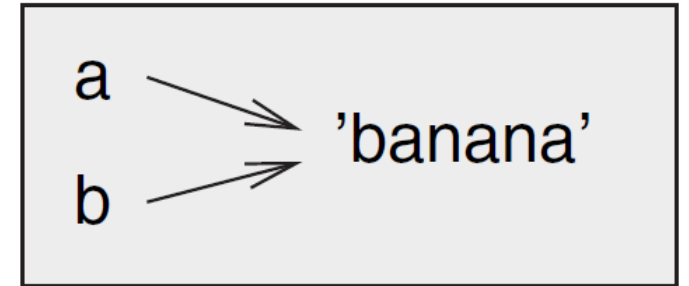
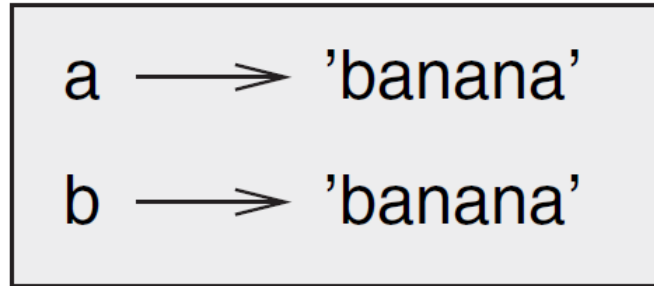


# 'is' OPERATOR

Given two strings

a = 'banana'

b = 'banana'



We know that a and b both refer to a string.

Are they referring to the same string?



# 'is' OPERATOR

```
>>> a = 'banana'
```

```
>>> b = 'banana'
```

```
>>> a == b
```

True



# 'is' OPERATOR

```
>>> a = 'banana'
```

```
>>> b = 'banana'
```

```
>>> a is b
```

True



# 'is' OPERATOR

```
>>> a = [1,2]
```

```
>>> b = [1,2]
```

```
>>> a == b
```

True



# 'is' OPERATOR

```
>>> a = [1,2]
```

```
>>> b = [1,2]
```

```
>>> a is b
```

False



# DEBUGGING

```
t = [1,2]
```

```
# add element 4 to list t
```

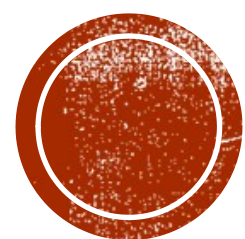
```
t.append([4])
```

```
t = t.append(4)
```

```
t + [4]
```

```
t = t + 4
```





DICTIONARY



# DICTIONARY

- Dictionary is like a list, but more general.
- In a list, the indices have to be integers; in a dictionary they can be (almost) any type.



# DICTIONARY

A dictionary is a mapping between  
KEYS and set of Values.

**key-value pair**



# DICTIONARY

- Lists index their entries based on the position in the list.
- Dictionaries are like bags - no order
- So we index the things we put in the dictionary with a “lookup tag”



# DICTIONARY

```
>>> eng2sp = {} )
```

```
>>> print(eng2sp)
```

```
{}
```



# DICTIONARY

```
>>> eng2sp['one'] = 'uno'
```

```
>>> print(eng2sp)
```

```
{'one': 'uno'}
```



# DICTIONARY

```
>>> eng2sp = {'one': 'uno', 'two': 'dos',  
'three': 'tres'}
```

```
>>> print(eng2sp)
```

```
{'three': 'tres', 'two': 'dos', 'one': 'uno'}
```



# DICTIONARY

The order of the key-value pairs is not the same. In fact, if you type the same example on your computer, you might get a different result. In general, the order of items in a dictionary is unpredictable.



# DICTIONARY

```
>>> item_price = {}
```

```
>>> item_price['milk'] = 10
```

```
>>> print(item_price['milk'])
```

```
10
```



# DICTIONARY

```
>>> item_price = {}
```

```
>>> item_price['milk'] = 10
```

```
>>> print(item_price['sugar'])
```

KeyError: 'sugar'



# DICTIONARY

```
>>> number_of_days = {'January':  
31, 'February': 28, 'March': 31}
```

```
>>> number_of_days['February']
```



# DEBUGGING

## Key Error

Traceback (most recent call last):

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

KeyError: 3



# ITERATION ON DICTIONARIES

Even though **dictionaries** are not stored in order, we can write a **for** loop that goes through all the **entries** in a **dictionary** - actually it goes through all of the **keys** in the **dictionary** and **looks up** the values



# ITERATION

```
eng2sp = {'one': 'uno', 'two': 'dos',  
          'three': 'tres'}
```

```
for k in eng2sp:  
    print(k)
```



# ITERATION: VALUES

```
eng2sp = {'one': 'uno', 'two': 'dos',  
'three': 'tres'}
```

```
for k in eng2sp:  
    print(eng2sp[k])
```



# 'in' OPERATOR

```
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three':  
'tres'}
```

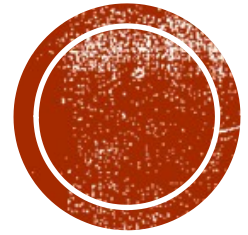
```
>>> 'one' in eng2sp
```

```
True
```

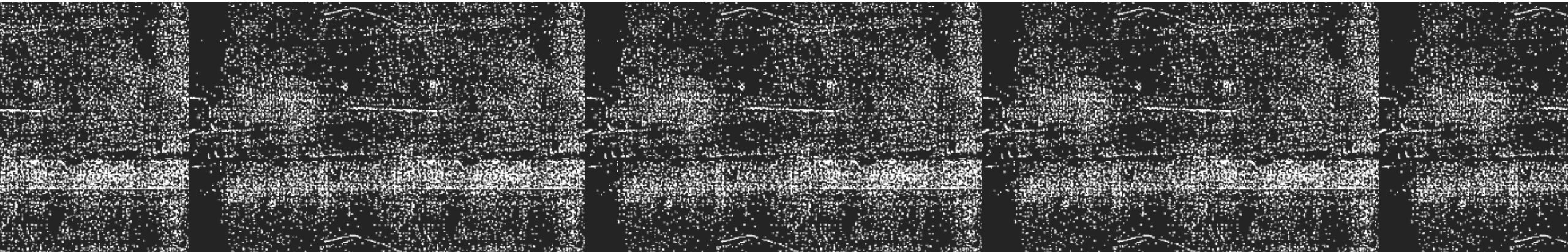
```
>>> 'uno' in eng2sp
```

```
False
```





# DEMO: CHARACTER COUNTER (HISTOGRAM)



# WHAT DATA TYPES CAN NOT BE KEYS



# RETRIEVING LISTS OF KEYS AND VALUES

```
>>> counts.keys()
```

```
>>> counts.values()
```

```
>>> counts.items()
```

