Part of the “science” in computer science is the design and use of _data structures and algorithms_.

An *algorithm* is a recipe, a procedure to solve a problem.

Every-day algorithms
- Radix sort: sorting a deck of cards
- One queue vs. many queues (ATM)
- First-fit vs. best-fit (parking)
- Evolution: survival of the fittest
- Arithmetic: long division
- MP3: compression
Data structures are particular ways of storing data to be used by an algorithm.

Roughly two kinds of data structures:

- *built-in* data structures; they are so common that they are provided by default.
- *user-defined* data structures (classes in object oriented programming) that are designed for a particular task.
Python built-in data structures:
- lists
- tuples
- strings
- dictionaries
- sets
- others...

list: an ordered sequence of items
contrast with strings.
List: similarities with strings

- Concatenate: + (but only of lists)
- Repeat: *
- Indexing (the [ ] operator)
- Slicing ([:])
- Membership (the in operator)
- Length (the len function)

List: difference from strings

- lists: a mixture of any Python objects
  - strings: only characters
- lists are mutable: can be changed
  - strings are immutable
- lists are designated with [ ], with elements separated by commas
  - strings are designated by quotes “”; no commas
Methods vs. Functions

Both are small programs.

- a function (such as `len`) that takes some arguments, the stuff in the parenthesis, and returns some value.

- a method is called in a special way: the “dot call”.
  It is called in the context of an object.
  Some return a value; some have side effects.

```python
my_list = ['a', 1, True]
my_list.append('z')
```

- the object that we are calling the method with
- the name of the method
- arguments to the method
A list is *mutable* so it can change:
- `my_list[0] = 'a'`  # index assignment
- `my_list.append()`, `my_list.extend()`
- `my_list.pop()`
- `my_list.insert()`, `my_list.remove()`
- `my_list.sort()`
- `my_list.reverse()`

**Sorting**
- Method changes the list, nothing returned
  `my_list.sort()`
- Function returns a new list, original list is unchanged
  `new_list = sorted(my_list)`
Mutable: you can change the value at some index in a list.

```python
my_list = ['a', 1, True, 3.14159]
my_list[0] = 'abc'
print(my_list)
['abc', 1, True, 3.14159]
```

You can even change an entire slice:

```python
my_list = ['a', 1, True, 3.14159]
my_list[0:2] = 'xyz'
print(my_list)
['x', 'y', 'z', True, 3.14159]
```
More slice assignment

- The item on the RHS must be iterable (a sequence that you can “walk” through)
- If you use the “stride” part of a slice, the number of items on the LHS must match the number of items on the RHS

Mutable operations

```python
my_list=[1,2,3,4]
my_list.append(27)
print(my_list) => [1,2,3,4,27]
```

Side effect: notice that there was NO RETURN VALUE from a mutable operation! The list was changed, but nothing was returned.
More mutables

my_list = ['abc', 'def', 'ghi', 'jkl']
print(my_list.insert(2, 'xyz'))
   ERROR, insert changes the list,
   but there is no return value, nothing to print.

print(my_list)
['abc', 'def', 'xyz', 'ghi', 'jkl']

Worksheet
L1 = [1,2]
L2 = ['a','b',L1]
print(L2)
    ['a','b',[1,2]]
L1[0] = 99
print(L2)
    ['a','b',[99,2]]
for works as with strings

```python
my_list = [1, 2, 3, 4]
for element in my_list:
    print(element, ',', element**2)
```

prints out first four squares.

Example forExamples.py
a perfect number

- Numbers and their factors were mysterious to the Greeks and early mathematicians.
- They were curious about the properties of numbers as they held some significance.
- A perfect number is a number whose sum of factors (excluding the number) equals the number.
- First perfect number is: 6 (1+2+3)

abundant, deficient

- abundant numbers summed to more than the number.
  - 12: 1+2+3+4+6 = 16
- deficient numbers summed to less than the number.
  - 13: 1
design

1. input a max
2. go through the numbers from 1 to max
   a) for each number, collect all the factors
   b) sum up the factors
   c) compare the sum and the number, and classify accordingly
3. summarize results

Tuples
Tuples are simply immutable lists.

Designated with (,)

my_tuple = (1,'a',3.14,True)

Why have an immutable list, a tuple, as a separate type?

You cannot accidentally change a tuple.
Everything that works with lists works with tuples except methods that modify the tuple.

Thus indexing, slicing, len, print all work as before. However, none of the mutable methods work: append, extend, del

Commas make a tuple

For tuples, you can think of a comma as the operator that makes a tuple, where the ( ) simply acts as a grouping:

```python
my_tuple = 1,2  # creates (1,2)
my_tuple = (1,) # creates (1)
my_tuple = (1)  # creates 1 not (1)
my_tuple = 1,   # creates (1)
```
Example
simpleTuples