Roadmap

Last week we learned

- Local variables
- Global variables
Roadmap

Last week we learned

- Local variables
- Global variables

This week we will learn

- Sequences
  - Lists
  - Strings
  - Tuples
Here is a table of Olympic medals from the 2014 Sochi Winter Games:

<table>
<thead>
<tr>
<th>Country</th>
<th>Gold</th>
<th>Silver</th>
<th>Bronze</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Austria</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Belarus</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Canada</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>China</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Croatia</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Czech Republic</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Finland</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>France</td>
<td>4</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Germany</td>
<td>8</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Great Britain</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Italy</td>
<td>0</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Japan</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Kazakhstan</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Latvia</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Netherlands</td>
<td>8</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>Norway</td>
<td>11</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Poland</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Russia</td>
<td>13</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>Slovakia</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Slovenia</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>South Korea</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Sweden</td>
<td>2</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Switzerland</td>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Ukraine</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>United States</td>
<td>9</td>
<td>7</td>
<td>12</td>
</tr>
</tbody>
</table>
Here is a table of Olympic medals from the 2014 Sochi Winter Games:

<table>
<thead>
<tr>
<th>Country</th>
<th>Gold</th>
<th>Silver</th>
<th>Bronze</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Austria</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Belarus</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Canada</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>China</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Croatia</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Czech Republic</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Finland</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>France</td>
<td>4</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Germany</td>
<td>8</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Great Britain</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Italy</td>
<td>0</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Japan</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Kazakhstan</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Latvia</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Netherlands</td>
<td>8</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>Norway</td>
<td>11</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Poland</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Russia</td>
<td>13</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>Slovakia</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Slovenia</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>South Korea</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Sweden</td>
<td>2</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Switzerland</td>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Ukraine</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>United States</td>
<td>9</td>
<td>7</td>
<td>12</td>
</tr>
</tbody>
</table>

How can we store this much data in Python? We would need $4 \times 26$ variables . . .
Here is a table of Olympic medals from the 2014 Sochi Winter Games:

<table>
<thead>
<tr>
<th>Country</th>
<th>Gold</th>
<th>Silver</th>
<th>Bronze</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Austria</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Belarus</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Canada</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>China</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Croatia</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Czech Republic</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Finland</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>France</td>
<td>4</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Germany</td>
<td>8</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Great Britain</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Italy</td>
<td>0</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Japan</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Kazakhstan</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Latvia</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Netherlands</td>
<td>8</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>Norway</td>
<td>11</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Poland</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Russia</td>
<td>13</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>Slovakia</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Slovenia</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>South Korea</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Sweden</td>
<td>2</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Switzerland</td>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Ukraine</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>United States</td>
<td>9</td>
<td>7</td>
<td>12</td>
</tr>
</tbody>
</table>

How can we store this much data in Python? We would need $4 \times 26$ variables . . .

The solution is to store all values together in a list.
Lists

To create a list, enclose the values in square brackets:

```python
>>> countries = [ "Australia", ... , "United States" ]
```

```python
>>> gold = [0, 4, 5, 10, 3, 0, 2, 1, 4, 8, 1, 0, 1, 0, 0, 8, 11, 4, 13, 1, 2, 3, 2, 6, 1, 9]
```
Lists

To create a list, enclose the values in square brackets:

```python
>>> countries = [ "Australia", ... , "United States" ]
>>> gold = [0, 4, 5, 10, 3, 0, 2, 1, 4, 8, 1, 0, 1, 0, 0,
   8, 11, 4, 13, 1, 2, 3, 2, 6, 1, 9]
```

A list is an object of type `list`. 
Lists

To create a list, enclose the values in square brackets:

```
>>> countries = [ "Australia", ... , "United States" ]
>>> gold = [0, 4, 5, 10, 3, 0, 2, 1, 4, 8, 1, 0, 1, 0, 0, 8, 11, 4, 13, 1, 2, 3, 2, 6, 1, 9]
```

A list is an object of type **list**.
We can access the elements of a list using an integer index.
The first element is at index **0**, the second at index **1**, and so on:

```
>>> countries[0]
'Australia'
>>> countries[21]
'South Korea'
>>> gold[21]
3
```
Lists

To create a list, enclose the values in square brackets:

```python
>>> countries = [ "Australia", ..., "United States" ]
>>> gold = [0, 4, 5, 10, 3, 0, 2, 1, 4, 8, 1, 0, 1, 0, 0,
          8, 11, 4, 13, 1, 2, 3, 2, 6, 1, 9]
```

A list is an object of type `list`.

We can access the elements of a list using an integer index.
The first element is at index 0, the second at index 1, and so on:

```python
>>> countries[0]
'Australia'
>>> countries[21]
'South Korea'
>>> gold[21]
3
```

Negative indices start at the end of the list:

```python
>>> countries[-1]
'United States'
>>> countries[-5]
'South Korea'
```
Lists

The length of a list is given by \texttt{len}:

```python
>>> len(countries)
26
```
Lists

The length of a list is given by `len`:

```python
>>> len(countries)
26
```

The empty list is written `[]` and has length zero.
Lists

The length of a list is given by `len`:

```python
>>> len(countries)
26
```

The empty list is written `[]` and has length zero.

Lists can contain a mixture of objects of any type:

```python
>>> korea = [ 'Korea', 'KR', 3, 3, 2 ]
>>> korea[1]
'KR'
>>> korea[2]
3
```
Lists

The length of a list is given by `len`:

```python
>>> len(countries)
26
```

The empty list is written `[]` and has length zero.

Lists can contain a mixture of objects of any type:

```python
>>> korea = [ 'Korea', 'KR', 3, 3, 2 ]
>>> korea[1]
'KR'
>>> korea[2]
3
```

Or even:

```python
>>> korea = [ "Korea", 'KR', (3, 3, 2) ]
```
Lists are mutable

A list of noble gases:

```python
>>> nobles = ['helium', 'none', 'argon', 'krypton', 'xenon']
```
Lists are mutable

A list of noble gases:

```python
>>> nobles = [ 'helium', 'none', 'argon', 'krypton',
            'xenon' ]
```

Oops. Correct the typo:

```python
>>> nobles[1] = "neon"
```
```python
>>> nobles
['helium', 'neon', 'argon', 'krypton', 'xenon']
```
Lists are mutable

A list of noble gases:

```python
>>> nobles = [ 'helium', 'none', 'argon', 'krypton', 'xenon' ]
```

Oops. Correct the typo:

```python
>>> nobles[1] = "neon"
```

```python
>>> nobles
['helium', 'neon', 'argon', 'krypton', 'xenon']
```

Oops oops. I forgot radon!

```python
>>> nobles.append('radon')
```

```python
>>> nobles
['helium', 'neon', 'argon', 'krypton', 'xenon', 'radon']
```
Aliasing

Reminder: An object can have more than one name. This is called aliasing. We have to be careful when working with mutable objects:
Aliasing

Reminder: An object can have more than one name. This is called **aliasing**. We have to be careful when working with mutable objects:

```python
>>> list1 = ['A', 'B', 'C']
>>> list2 = list1
>>> len(list1)
3
>>> list2.append('D')
>>> len(list1)
4
>>> list1[1] = 'X'
>>> list2
['A', 'X', 'C', 'D']
```
Aliasing

Reminder: An object can have more than one name. This is called aliasing. We have to be careful when working with mutable objects:

```python
>>> list1 = ['A', 'B', 'C']
>>> list2 = list1
>>> len(list1)
3
>>> list2.append('D')
>>> len(list1)
4
>>> list1[1] = 'X'
>>> list2
['A', 'X', 'C', 'D']
```

```python
>>> list1 = ['A', 'B', 'C']
>>> list2 = list1
>>> len(list1)
3
>>> list2.append('D')
>>> len(list1)
3
>>> list1[1] = 'X'
>>> list2
['A', 'B', 'C', 'D']
```
Aliasing

Reminder: An object can have more than one name. This is called **aliasing**. We have to be careful when working with mutable objects:

```python
>>> list1 = ['A', 'B', 'C']
>>> list2 = list1
>>> len(list1)
3
>>> list2.append('D')
>>> len(list1)
4
>>> list1[1] = 'X'
>>> list2
['A', 'X', 'C', 'D']
>>> list1 is list2
True
```

```python
>>> list1 = ['A', 'B', 'C']
>>> list2 = ['A', 'B', 'C']
>>> len(list1)
3
>>> list2.append('D')
>>> len(list1)
3
>>> list1[1] = 'X'
>>> list2
['A', 'B', 'C', 'D']
```
Aliasing

Reminder: An object can have more than one name. This is called **aliasing**. We have to be careful when working with mutable objects:

```python
>>> list1 = ["A","B","C"]
>>> list2 = list1
>>> len(list1)
3
>>> list2.append("D")
>>> len(list1)
4
>>> list1[1] = "X"
>>> list2
['A', 'X', 'C', 'D']
>>> list1 is list2
True
```

```python
>>> list1 = ["A","B","C"]
>>> list2 = ["A","B","C"]
>>> len(list1)
3
>>> list2.append("D")
>>> len(list1)
3
>>> list1[1] = "X"
>>> list2
['A', 'B', 'C', 'D']
>>> list1 is list2
False
```
Reminder: Objects with two names

The same object can have more than one name:

hubo = Robot("yellow")
Reminder: Objects with two names

The same object can have more than one name:

hubo = Robot("yellow")
Reminder: Objects with two names

The same object can have more than one name:

hubo = Robot("yellow")
hubo.move()
ami = hubo
Reminder: Objects with two names

The same object can have more than one name:

hubo = Robot("yellow")
hubo.move()
ami = hubo
ami.turn_left()
hubo.move()
Reminder: Objects with two names

The same object can have more than one name:

```python
hubo = Robot("yellow")
hubo.move()
ami = hubo
ami.turn_left()
hubo.move()

hubo = Robot("blue")
hubo.move()
ami.turn_left()
ami.move()
```
Built-in functions on lists

`len` returns length of a list.

`sum` the sum of the elements.

`max` the largest element, `min` the smallest element:

```python
>>> len(gold), sum(gold), max(gold), min(gold)
(26, 99, 13, 0)

>>> len(silver), sum(silver), max(silver)
(26, 97, 11)

>>> len(bronze), sum(bronze), max(bronze)
(26, 99, 12)
```
Traversing a list

A **for** loop looks at every element of a list:

```python
for country in countries:
    print(country)
```
Traversing a list

A `for` loop looks at every element of a list:

```python
for country in countries:
    print(country)
```

We can get a range object from the `range` function as below:

```python
>>> range(10)
range(0, 10)
>>> type(range(10))
<class 'range'>
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(10, 15))
[10, 11, 12, 13, 14]
```
Traversing a list

A `for` loop looks at every element of a list:

```python
for country in countries:
    print(country)
```

We can get a range object from the `range` function as below:

```python
>>> range(10)
range(0, 10)
>>> type(range(10))
<class 'range'>
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(10, 15))
[10, 11, 12, 13, 14]
```

If we want to modify elements, we need the index:

```python
>>> l = list(range(1, 11))
>>> for i in range(len(l)):
...     l[i] = l[i] ** 2
>>> l
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```
Traversing several lists

Let’s print out the total number of medals for each country:

```python
>>> for i in range(len(countries)):
...     print(countries[i], gold[i]+silver[i]+bronze[i])
```
Traversing several lists

Let’s print out the total number of medals for each country:

```python
>>> for i in range(len(countries)):
...     print(countries[i], gold[i]+silver[i]+bronze[i])
```

We can create a new list:

```python
>>> totals = []
>>> for i in range(len(countries)):
...     medals = gold[i]+silver[i]+bronze[i]
...     totals.append( (medals, countries[i]) )
```
Traversing several lists

Let’s print out the total number of medals for each country:

```python
>>> for i in range(len(countries)):
...    print(countries[i], gold[i]+silver[i]+bronze[i])
```

We can create a new list:

```python
>>> totals = []
>>> for i in range(len(countries)):
...    medals = gold[i]+silver[i]+bronze[i]
...    totals.append((medals, countries[i]))
```

The list `totals` is now a list of tuples `(medals, country)`.  

```python
>>> totals
[(3, 'Australia'), (17, 'Austria'), (6, 'Belarus'), ... (4, 'Latvia'), (24, 'Netherlands'), ...,
(8, 'South Korea'), ... (2, 'Ukraine'), (28,  
'United States')]
```
We can sort a list using its `sort` method:

```python
>>> ta = [ "JinYeong", "Jeongmin", "Minsuk",
       "Dohoo", "Sangjae", "Byung-Jun" ]
>>> ta.sort()
>>> ta
['Byung-Jun', 'Dohoo', 'Jeongmin', 'JinYeong',
 'Minsuk', 'Sangjae']
We can sort a list using its `sort` method:

```python
>>> ta.sort()
>>> ta
['Byung-Jun', 'Dohoo', 'Jeongmin', 'JinYeong', 'Minsuk', 'Sangjae']
```

Let's sort the medal totals: `totals.sort()`.

```python
>>> totals.sort()
>>> totals
[(1, 'Croatia'), (1, 'Kazakhstan'), (1, 'Slovakia'), (2, 'Ukraine'), (3, 'Australia'), ..., (8, 'Japan'), (8, 'Slovenia'), (8, 'South Korea'), ...
```

... (33, 'Russia')]

We rather want the countries with the largest number of medals at the top:

```python
>>> totals.reverse()
>>> totals
[(33, 'Russia'), (28, 'United States'), ..., (8, 'South Korea'), ..., (1, 'Kazakhstan'), (1, 'Croatia')]
```
Reversing

We rather want the countries with the largest number of medals at the top:

```python
>>> totals.reverse()
>>> totals
[(33, 'Russia'), (28, 'United States'), ..., (8, 'South Korea'), ..., (1, 'Kazakhstan'), (1, 'Croatia')]
```

Actually we only care about the top 10:

```python
>>> top_ten = totals[:10]
>>> for p in top_ten:
...    medals, country = p
...    print(medals, country)
```

13 / 22
We rather want the countries with the largest number of medals at the top:

```python
>>> totals.reverse()
>>> totals
[(33, 'Russia'), (28, 'United States'), ..., (8, 'South Korea'), ..., (1, 'Kazakhstan'), (1, 'Croatia')]
```

Actually we only care about the top 10:

```python
>>> top_ten = totals[:10]
>>> for p in top_ten:
...    medals, country = p
...    print(medals, country)
```

We can unpack the elements in a list immediately

```python
>>> for medals, country in top_ten:
...    print(medals, country)
```
Slicing

Slicing creates a **new list** with elements of the given list:

\[
\text{sublist} = \text{mylist}[i:j]
\]

Then **sublist** contains elements \(i, \ i+1, \ldots j-1\) of mylist.
Slicing

Slicing creates a **new list** with elements of the given list:

```
sublist = mylist[i:j]
```

Then `sublist` contains elements `i, i+1, ... j-1` of `mylist`.

If `i` is omitted, the sublist starts with the first element.
Slicing

Slicing creates a new list with elements of the given list:

\[
\text{sublist} = \text{mylist}[i:j]
\]

Then \text{sublist} contains elements \(i, \ i+1, \ldots \ j-1\) of mylist.

If \(i\) is omitted, the sublist starts with the first element.

If \(j\) is omitted, then the sublist ends with the last element.
Slicing creates a **new list** with elements of the given list:

```python
sublist = mylist[i:j]
```

Then `sublist` contains elements $i$, $i+1$, $\ldots$ $j-1$ of `mylist`.

If $i$ is omitted, the sublist starts with the first element.

If $j$ is omitted, then the sublist ends with the last element.

**Special case:** We can create a copy of a list with

```python
list2 = list1[:]
```
Let’s create the top-10 lexicographical ranking:

```python
table = []
for i in range(len(countries)):
    table.append((gold[i], silver[i], bronze[i], countries[i]))

table.sort()
top_ten = table[-10:]
top_ten.reverse()
for g, s, b, country in top_ten:
    print(country, g, s, b)
```
Let's create the top-10 lexicographical ranking:

```python
table = []
for i in range(len(countries)):
    table.append( (gold[i], silver[i], bronze[i], countries[i]) )
table.sort()
top_ten = table[-10:]
top_ten.reverse()
for g, s, b, country in top_ten:
    print(country, g, s, b)
```

<table>
<thead>
<tr>
<th>Country</th>
<th>Gold</th>
<th>Silver</th>
<th>Bronze</th>
</tr>
</thead>
<tbody>
<tr>
<td>Russia</td>
<td>13</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>Norway</td>
<td>11</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Canada</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>United States</td>
<td>9</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>Netherlands</td>
<td>8</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>Germany</td>
<td>8</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Switzerland</td>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Belarus</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Austria</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>France</td>
<td>4</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>
Let’s find all countries that have only one kind of medal (assuming that no country has 0 medal):

def no_medals(countries, al, bl):
    result = []
    for i in range(len(countries)):
        if al[i] == 0 and bl[i] == 0:
            result.append(countries[i])
    return result

only_gold = no_medals(countries, silver, bronze)
only_silver = no_medals(countries, gold, bronze)
only_bronze = no_medals(countries, gold, silver)

only_one = only_gold + only_silver + only_bronze
List methods

List objects \( L \) have the following methods:

- \( L.append(v) \) add object \( v \) at the end
- \( L.insert(i, v) \) insert element at position \( i \)
- \( L.pop() \) remove and return last element
- \( L.pop(i) \) remove and return element at position \( i \)
- \( L.remove(v) \) remove first element equal to \( v \)
- \( L.index(v) \) return index of first element equal to \( v \)
- \( L.count(v) \) return number of elements equal to \( v \)
- \( L.extend(K) \) append all elements of sequence \( K \) to \( L \)
- \( L.reverse() \) reverse the list
- \( L.sort() \) sort the list
List methods

List objects \( L \) have the following methods:

- \( L.append(v) \) add object \( v \) at the end
- \( L.insert(i, v) \) insert element at position \( i \)
- \( L.pop() \) remove and return last element
- \( L.pop(i) \) remove and return element at position \( i \)
- \( L.remove(v) \) remove first element equal to \( v \)
- \( L.index(v) \) return index of first element equal to \( v \)
- \( L.count(v) \) return number of elements equal to \( v \)
- \( L.extend(K) \) append all elements of sequence \( K \) to \( L \)
- \( L.reverse() \) reverse the list
- \( L.sort() \) sort the list

What is the difference?

\[
L.append(13) \\
L + [ 13 ]
\]
Sequences

Lists are a kind of sequence. We already met other kinds of sequences: strings and tuples:
Sequences

Lists are a kind of sequence. We already met other kinds of sequences: strings and tuples:

Strings:

```python
>>> a = "CS101"
>>> a[-1]
'1'
>>> a[2:]
'101'
>>> for i in a:
...   print (i)
C
S
1
0
1
```
Sequences

Lists are a kind of sequence. We already met other kinds of sequences: strings and tuples:

Strings:

```python
>>> a = "CS101"
>>> a[-1]
'1'
>>> a[2:]
'101'
>>> for i in a:
...    print (i)
CS101
```

Tuples:

```python
>>> t = ("CS101", "A+", 13)
>>> t[0]
'CS101'
>>> t[-1]
13
>>> t[1:]
('A+', 13)
>>> for i in t:
...    print (i)
CS101
A+
13
```
Lists, tuples, strings

Lists and tuples are very similar, but lists are **mutable**, while tuples (and strings) are **immutable**:

```python
>>> t[0] = "CS206"
TypeError: 'tuple' object does not support item assignment
```
Lists, tuples, strings

Lists and tuples are very similar, but lists are **mutable**, while tuples (and strings) are **immutable**:

```python
>>> t[0] = "CS206"
TypeError: 'tuple' object does not support item assignment
```

We can convert a sequence into a list or tuple using the `list` and `tuple` functions:

```python
>>> list(t)
['CS101', 'A+', 13]
>>> tuple(gold)
(0, 4, 5, 10, 3, 0, 2, 1, 4, ..., 2, 6, 1, 9)
>>> list("CS101")
['C', 'S', '1', '0', '1']
```
Using four lists to store the medal information is not typical for Python. We would normally make a single list of tuples:

```python
medals = [ ( 'Australia', 0, 2, 1 ),
           ( 'Austria', 4, 8, 5 ),
           ...
           ( 'United States', 9, 7, 12 ) ]
```
Using four lists to store the medal information is not typical for Python. We would normally make a single list of tuples:

```
medals = [ ('Australia', 0, 2, 1 ),
           ('Austria', 4, 8, 5 ),
           ...
           ('United States', 9, 7, 12 ) ]
```

Print the total number of medals for each country:

```
def print_totals1():
    for country, g, s, b in medals:
        print(country + "::", g + s + b)
```
Using four lists to store the medal information is not typical for Python. We would normally make a single list of tuples:

```python
totalMedals = [ ('Australia', 0, 2, 1 ),
                ('Austria', 4, 8, 5 ),
                ...
                ('United States', 9, 7, 12 ) ]
```

Print the total number of medals for each country:

```python
def print_totals1():
    for country, g, s, b in totalMedals:
        print(country + " : ", g + s + b)

def print_totals2():
    for item in totalMedals:
        print(item[0] + " : ", sum(item[1:]))
```
We want to create a histogram of medals:

```python
def histogram():
    t = [0] * 13
    for item in medals:
        total = sum(item[1:])
        t[total // 3] += 1
    for i in range(13):
        print (str(3*i) + "~" + str(3*i+2) + "\t" + ("*" * t[i])
```

<table>
<thead>
<tr>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0~2:</td>
</tr>
<tr>
<td>3~5:</td>
</tr>
<tr>
<td>6~8:</td>
</tr>
<tr>
<td>9~11:</td>
</tr>
<tr>
<td>12~14:</td>
</tr>
<tr>
<td>15~17:</td>
</tr>
<tr>
<td>18~20:</td>
</tr>
<tr>
<td>21~23:</td>
</tr>
<tr>
<td>24~26:</td>
</tr>
<tr>
<td>27~29:</td>
</tr>
<tr>
<td>30~32:</td>
</tr>
<tr>
<td>33~35:</td>
</tr>
<tr>
<td>36~38:</td>
</tr>
</tbody>
</table>
Computing prime numbers

Sieve of Eratosthenes

Code

```python
def sieve(n):
    t = [2] + list(range(3, n+1, 2))
    sqrtn = int(math.sqrt(n))
    i = 0
    while t[i] <= sqrtn:
        # remove all multiples of t[i]
        p = t[i]
        for j in range(len(t)-1, i, -1):
            if t[j] % p == 0:
                t.pop(j)
        i += 1
    return t
```

Return list when parameter n is 100

\[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97\]