



Here is a table of olympic medals from the 2010 Vancouver winter games:

Source: www.vancouver2010.com

Australia	2	1	0
Austria	4	6	6
Belarus	1	1	1
Canada	14	7	5
China	5	2	4
Croatia	0	2	1
Czech Republic	2	0	4
Estonia	0	1	0
Finland	0	1	4
France	2	3	6
Germany	10	13	7
Great Britain	1	0	0
Italy	1	1	3
Japan	0	3	2
Kazakhstan	0	1	0
Korea	6	6	2
Latvia	0	2	0
Netherlands	4	1	3
Norway	9	8	6
Poland	1	3	2
Russian Federation	3	5	7
Slovakia	1	1	1
Slovenia	0	2	1
Sweden	5	2	4
Switzerland	6	0	3
United States	9	15	13



Here is a table of olympic medals from the 2010 Vancouver winter games:

Source: www.vancouver2010.com

Australia	2	1	0
Austria	4	6	6
Belarus	1	1	1
Canada	14	7	5
China	5	2	4
Croatia	0	2	1
Czech Republic	2	0	4
Estonia	0	1	0
Finland	0	1	4
France	2	3	6
Germany	10	13	7
Great Britain	1	0	0
Italy	1	1	3
Japan	0	3	2
Kazakhstan	0	1	0
Korea	6	6	2
Latvia	0	2	0
Netherlands	4	1	3
Norway	9	8	6
Poland	1	3	2
Russian Federation	3	5	7
Slovakia	1	1	1
Slovenia	0	2	1
Sweden	5	2	4
Switzerland	6	0	3
United States	9	15	13

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



Here is a table of olympic medals from the 2010 Vancouver winter games:

Source: www.vancouver2010.com

Australia	2	1	0
Austria	4	6	6
Belarus	1	1	1
Canada	14	7	5
China	5	2	4
Croatia	0	2	1
Czech Republic	2	0	4
Estonia	0	1	0
Finland	0	1	4
France	2	3	6
Germany	10	13	7
Great Britain	1	0	0
Italy	1	1	3
Japan	0	3	2
Kazakhstan	0	1	0
Korea	6	6	2
Latvia	0	2	0
Netherlands	4	1	3
Norway	9	8	6
Poland	1	3	2
Russian Federation	3	5	7
Slovakia	1	1	1
Slovenia	0	2	1
Sweden	5	2	4
Switzerland	6	0	3
United States	9	15	13

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

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



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

```
countries = [ "Australia", ... , "United States" ]  
gold = [2, 4, 1, 14, 5, 0, 2, 0, 0, 2, 10, 1, 1, 0,  
         0, 6, 0, 4, 9, 1, 3, 1, 0, 5, 6, 9]
```



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

```
countries = [ "Australia", ... , "United States" ]  
gold = [2, 4, 1, 14, 5, 0, 2, 0, 0, 2, 10, 1, 1, 0,  
         0, 6, 0, 4, 9, 1, 3, 1, 0, 5, 6, 9]
```

A list is an object of type `list`.



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

```
countries = [ "Australia", ... , "United States" ]  
gold = [2, 4, 1, 14, 5, 0, 2, 0, 0, 2, 10, 1, 1, 0,  
        0, 6, 0, 4, 9, 1, 3, 1, 0, 5, 6, 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[15]  
'Korea'  
>>> gold[15]  
6
```



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

```
countries = [ "Australia", ... , "United States" ]  
gold = [2, 4, 1, 14, 5, 0, 2, 0, 0, 2, 10, 1, 1, 0,  
        0, 6, 0, 4, 9, 1, 3, 1, 0, 5, 6, 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[15]  
'Korea'  
>>> gold[15]  
6
```

Negative indices start at the end of the list:

```
>>> countries[-1]  
'United States'  
>>> countries[-11]  
'Korea'
```



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

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




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

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

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



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

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

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

Lists can contain a mixture of objects of any type:

```
>>> korea = [ 'Korea', 'KR', 6, 6, 2 ]
>>> korea[1]
'KR'
>>> korea[2]
6
```



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

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

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

Lists can contain a mixture of objects of any type:

```
>>> korea = [ 'Korea', 'KR', 6, 6, 2 ]
>>> korea[1]
'KR'
>>> korea[2]
6
```

Or even:

```
>>> korea = [ "Korea", 'KR', ( 6, 6, 2) ]
```



A list of noble gases:

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



A list of noble gases:

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

Oops. Correct the typo:

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

```
>>> nobles
```

```
['helium', 'neon', 'argon', 'krypton', 'xenon']
```



A list of noble gases:

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

Oops. Correct the typo:

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

```
>>> nobles
```

```
['helium', 'neon', 'argon', 'krypton', 'xenon']
```

Oops oops. I forgot radon!

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

```
>>> nobles
```

```
['helium', 'neon', 'argon', 'krypton', 'xenon', 'radon']
```



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

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



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

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

```
>>> 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']
```




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

```
>>> 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
```

```
>>> 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
```



`len` returns length of a list, `sum` the sum of the elements, `max` the largest element, `min` the smallest element:

```
>>> len(gold), sum(gold), max(gold), min(gold)
(26, 86, 14, 0)
```

```
>>> len(silver), sum(silver), max(silver)
(26, 87, 15)
```

```
>>> len(bronze), sum(bronze), max(bronze)
(26, 85, 13)
```



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

```
for country in countries:  
    print country
```



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

```
for country in countries:  
    print country
```

The **range** function returns a list:

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



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

```
for country in countries:  
    print country
```

The **range** function returns a list:

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

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

```
>>> l = 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]
```



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

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



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

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

We can create a new list:

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



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

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

We can create a new list:

```
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**).

```
[(3, 'Australia'), (16, 'Austria'), (3, 'Belarus'), ...,
(14, 'Korea'), (2, 'Latvia'), (8, 'Netherlands'), (23,
'Norway'), (6, 'Poland'), (15, 'Russian Federation'), (3,
'Slovakia'), (3, 'Slovenia'), (11, 'Sweden'), (9,
'Switzerland'), (37, 'United States')]
```




We can sort a list using its **sort** method:

```
>>> ta = [ "Minsu", "Hyunik", "Hyo-Sil",  
...       "Junghwan", "YeongJae", "Jinki" ]  
>>> ta.sort()  
>>> ta  
['Hyo-Sil', 'Hyunik', 'Jinki', 'Junghwan',  
'Minsu', 'YeongJae']
```



We can sort a list using its `sort` method:

```
>>> ta = [ "Minsu", "Hyunik", "Hyo-Sil",  
...       "Junghwan", "YeongJae", "Jinki" ]  
>>> ta.sort()  
>>> ta  
['Hyo-Sil', 'Hyunik', 'Jinki', 'Junghwan',  
'Minsu', 'YeongJae']
```

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

```
[(1, 'Estonia'), (1, 'Great Britain'), (1, 'Kazakhstan'),  
(2, 'Latvia'), (3, 'Australia'), (3, 'Belarus'), (3,  
'Croatia'), (3, 'Slovakia'), (3, 'Slovenia'),... (11,  
'Sweden'), (14, 'Korea'), (15, 'Russian Federation'), (16,  
'Austria'), (23, 'Norway'), (26, 'Canada'), (30,  
'Germany'), (37, 'United States')]
```



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

```
totals.reverse()
```

```
[(37, 'United States'), (30, 'Germany'), (26, 'Canada'),  
(23, 'Norway'), (16, 'Austria'), (15, 'Russian  
Federation'), (14, 'Korea'), (11, 'Sweden'), ... (1,  
'Estonia')]
```



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

```
totals.reverse()
```

```
[(37, 'United States'), (30, 'Germany'), (26, 'Canada'),  
(23, 'Norway'), (16, 'Austria'), (15, 'Russian  
Federation'), (14, 'Korea'), (11, 'Sweden'), ... (1,  
'Estonia')]
```

Actually we only care about the top-ten:

```
top_ten = totals[:10]  
for p in top_ten:  
    medals, country = p  
    print medals, country
```



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

```
totals.reverse()
```

```
[(37, 'United States'), (30, 'Germany'), (26, 'Canada'),  
(23, 'Norway'), (16, 'Austria'), (15, 'Russian  
Federation'), (14, 'Korea'), (11, 'Sweden'), ... (1,  
'Estonia')]
```

Actually we only care about the top-ten:

```
top_ten = totals[:10]
```

```
for p in top_ten:
```

```
    medals, country = p
```

```
    print medals, country
```

Slicing



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

```
totals.reverse()
```

```
[(37, 'United States'), (30, 'Germany'), (26, 'Canada'),  
(23, 'Norway'), (16, 'Austria'), (15, 'Russian  
Federation'), (14, 'Korea'), (11, 'Sweden'), ... (1,  
'Estonia')]
```

Actually we only care about the top-ten:

```
top_ten = totals[:10]
```

```
for p in top_ten:
```

```
    medals, country = p
```

```
    print medals, country
```

```
for medals, country in top_ten:
```

```
    print medals, country
```

Slicing

Unpack immediately



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

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

Then **sublist** contains elements $i, i + 1, \dots, j - 1$ of **mylist**.



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

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

Then **sublist** contains elements $i, i + 1, \dots, j - 1$ of **mylist**.

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



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

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

Then **sublist** contains elements $i, i + 1, \dots, j - 1$ 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:

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

Then **sublist** contains elements $i, i + 1, \dots, j - 1$ 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

```
list2 = list1[:]
```



Let's create the top-10 lexicographical ranking:

```
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:

```
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
```

```
Canada 14 7 5
Germany 10 13 7
United States 9 15 13
Norway 9 8 6
Korea 6 6 2
Switzerland 6 0 3
Sweden 5 2 4
China 5 2 4
Austria 4 6 6
Netherlands 4 1 3
```



Let's find all countries that have only one kind of medal:

```
def no_medals(countries, a1, b1):  
    result = []  
    for i in range(len(countries)):  
        if a1[i] == 0 and b1[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
```



Let's find all countries that have only one kind of medal:

```
def no_medals(countries, a1, b1):  
    result = []  
    for i in range(len(countries)):  
        if a1[i] == 0 and b1[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 concatenation



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 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 ]
```




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



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

Strings:

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



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

Strings:

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

Tuples:

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



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

```
>>> t[0] = "CS206"
```

```
TypeError: 'tuple' object does not support  
item assignment
```



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

```
>>> 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:

```
>>> list(t)
```

```
['CS101', 'A+', 13]
```

```
>>> tuple(gold)
```

```
(2, 4, 1, 14, 5, 0, 2, 0, 0, ..., 0, 5, 6, 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:

```
medals = [ ( 'Australia', 2, 1, 0 ),  
            ( 'Austria', 4, 6, 6 ),  
            ...  
            ( 'United States', 9, 15, 13 ) ]
```



Using four lists to store the medal information is not typical for Python. We would normally make a single list of tuples:

```
medals = [ ( 'Australia', 2, 1, 0 ),  
            ( 'Austria', 4, 6, 6 ),  
            ...  
            ( 'United States', 9, 15, 13 ) ]
```

Print 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:

```
medals = [ ( 'Australia', 2, 1, 0 ),  
            ( 'Austria', 4, 6, 6 ),  
            ...  
            ( 'United States', 9, 15, 13 ) ]
```

Print total number of medals for each country:

```
def print_totals1():  
    for country, g, s, b in medals:  
        print country + ":", g + s + b  
  
def print_totals2():  
    for item in medals:  
        print item[0] + ":", sum(item[1:])
```




Instead of creating a new list, let's sort the original list by total number of medals:

```
def compare(item1, item2):  
    medals1 = sum(item1[1:])  
    medals2 = sum(item2[1:])  
    return cmp(medals2, medals1)
```

```
def top_ten():  
    medals.sort(compare)  
    top_ten = medals[:10]  
    for item in top_ten:  
        print item[0] + ":", sum(item[1:])
```

```
United States: 37  
Germany: 30  
Canada: 26  
Norway: 23  
Austria: 16  
Russian Federation: 15  
Korea: 14  
China: 11  
France: 11  
Sweden: 11
```



Instead of creating a new list, let's sort the original list by total number of medals:

```
def compare(item1, item2):  
    medals1 = sum(item1[1:])  
    medals2 = sum(item2[1:])  
    return cmp(medals2, medals1)
```

```
def top_ten():  
    medals.sort(compare)  
    top_ten = medals[:10]  
    for item in top_ten:  
        print item[0] + ":", sum(item[1:])
```

United States: 37
Germany: 30
Canada: 26
Norway: 23
Austria: 16
Russian Federation: 15
Korea: 14
China: 11
France: 11
Sweden: 11

`cmp(a,b)` returns `-1` if $a < b$, `0` if $a = b$, and `+1` if $a > b$.



We want to create a histogram of medals:

```
0~2:    ****
3~5:    ****
6~8:    ***
9~11:   ****
12~14:  *
15~17:  **
18~20:
21~23:  *
24~26:  *
27~29:
30~32:  *
33~35:
36~38:  *
```



We want to create a histogram of medals:

```
0~2:    ****
3~5:    ****
6~8:    ***
9~11:   ****
12~14:  *
15~17:  **
18~20:
21~23:  *
24~26:  *
27~29:
30~32:  *
33~35:
36~38:  *
```

```
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])
```



Sieve of Eratosthenes

```
def sieve(n):
    t = range(3, n, 2)
    sqrt_n = int(math.sqrt(n))
    i = 0
    while t[i] <= sqrt_n:
        # 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
```



Sieve of Eratosthenes

```
def sieve(n):  
    t = range(3, n, 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
```

3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137,
139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199,
211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277,
281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359,
367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439,
443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521,
523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607,
613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683,
691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773,
787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863,
877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967,
971, 977, 983, 991, 997