CS101 - Sequences: Lists, Strings, and Tuples

Lecture 6

School of Computing
KAIST

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

Lots of data

Here is a table of Olympic medals from the 2014 Sochi Winter Games:

Australia	0	2	1
Austria	4	8	5
Belarus	5	0	1
Canada	10	10	5
China	3	4	2
Croatia	0	1	0
Czech Republic	2	4	2
Finland	1	3	1
France	4	4	7
Germany	8	6	5
Great Britain	1	1	2
Italy	0	2	6
Japan	1	4	3
Kazakhstan	0	0	1
Latvia	0	2	2
Netherlands	8	7	9
Norway	11	5	10
Poland	4	1	1
Russia	13	11	9
Slovakia	1	0	0
Slovenia	2	2	4
South Korea	3	3	2
Sweden	2	7	6
Switzerland	6	3	2
Ukraine	1	0	1
United States	9	7	12

Lots of data

Here is a table of Olympic medals from the 2014 Sochi Winter Games:

Australia	0	2	1
Austria	4	8	5
Belarus	5	0	1
Canada	10	10	5
China	3	4	2
Croatia	0	1	0
Czech Republic	2	4	2
Finland	1	3	1
France	4	4	7
Germany	8	6	5
Great Britain	1	1	2
Italy	0	2	6
Japan	1	4	3
Kazakhstan	0	0	1
Latvia	0	2	2
Netherlands	8	7	9
Norway	11	5	10
Poland	4	1	1
Russia	13	11	9
Slovakia	1	0	0
Slovenia	2	2	4
South Korea	3	3	2
Sweden	2	7	6
Switzerland	6	3	2
Ukraine	1	0	1
United States	9	7	12

How can we store this much data in Python?

We would need 4×26 variables . . .

Lots of data

Here is a table of Olympic medals from the 2014 Sochi Winter Games:

Australia	0	2	1
Austria	4	8	5
Belarus	5	0	1
Canada	10	10	5
China	3	4	2
Croatia	0	1	0
Czech Republic	2	4	2
Finland	1	3	1
France	4	4	7
Germany	8	6	5
Great Britain	1	1	2
Italy	0	2	6
Japan	1	4	3
Kazakhstan	0	0	1
Latvia	0	2	2
Netherlands	8	7	9
Norway	11	5	10
Poland	4	1	1
Russia	13	11	9
Slovakia	1	0	0
Slovenia	2	2	4
South Korea	3	3	2
Sweden	2	7	6
Switzerland	6	3	2
Ukraine	1	0	1
United States	9	7	12

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**.

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]
```

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**.

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:

```
>>> 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
```

Negative indices start at the end of the list:

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

Lists

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

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

Lists

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

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

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

Lists

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', 3, 3, 2 ]
>>> korea[1]
'KR'
>>> korea[2]
3
```

Lists

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', 3, 3, 2 ]
>>> korea[1]
'KR'
>>> korea[2]
3
```

Or even:

```
>>> korea = [ "Korea", 'KR', (3, 3, 2) ]
```

Lists are mutable

A list of noble gases:

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

Lists are mutable

A list of noble gases:

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

Oops. Correct the typo:

```
>>> nobles[1] = "neon"
>>> nobles
['helium', 'neon', 'argon', 'krypton', 'xenon']
```

Lists are mutable

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']
```

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:

```
>>> 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:

```
>>> 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']
```

Aliasing

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']
```

```
>>> list1 is list2
True
```

Aliasing

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
```

Reminder: Objects with two names

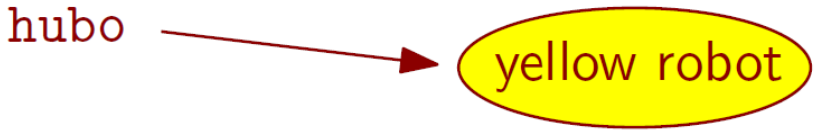
The same object can have more than one name:

```
hubo = Robot("yellow")
```

Reminder: Objects with two names

The same object can have more than one name:

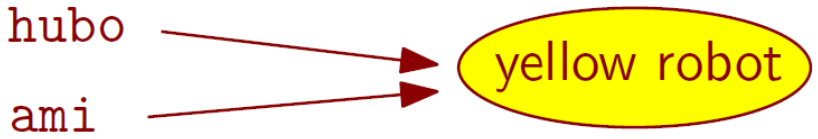
```
hubo = Robot("yellow")
```



Reminder: Objects with two names

The same object can have more than one name:

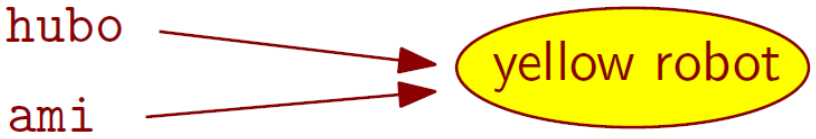
```
hubo = Robot("yellow")
hubo.move()
ami = hubo
```



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()
```

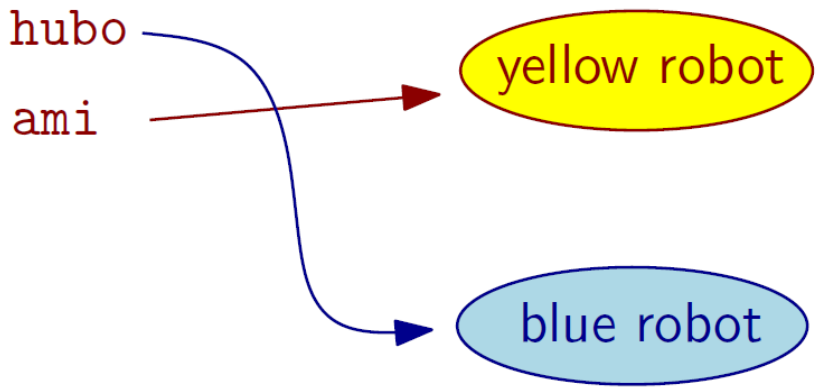


Reminder: Objects with two names

The same object can have more than one name:

```
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:

```
>>> 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:

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

Traversing a list

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

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

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

```
>>> 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:

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

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

```
>>> 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:

```
>>> 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:

```
>>> 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:

```
>>> 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]) )
```

Traversing several lists

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)**.

```
>>> totals
[(3, 'Australia'), (17, 'Austria'), (6, 'Belarus'),
 ↪  ..., (4, 'Latvia'), (24, 'Netherlands'), ...,
 ↪  (8, 'South Korea'), ..., (2, 'Ukraine'), (28,
 ↪  'United States')]
```

Sorting

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

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

Sorting

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

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

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

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

Reversing

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

```
>>> 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:

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

Actually we only care about the top 10:

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

Reversing

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

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

Actually we only care about the top 10:

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

We can unpack the elements in a list immediately

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

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.

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:

```
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.

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

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.

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[:]
```

Ranking

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)
```

Ranking

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)
```

```
Russia 13 11 9
Norway 11 5 10
Canada 10 10 5
United States 9 7 12
Netherlands 8 7 9
Germany 8 6 5
Switzerland 6 3 2
Belarus 5 0 1
Austria 4 8 5
France 4 4 7
```

Selecting elements

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:

```
>>> 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:

```
>>> a = "CS101"
>>> 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, tuples, strings

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, tuples, strings

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)
(0, 4, 5, 10, 3, 0, 2, 1, 4, ..., 2, 6, 1, 9)
>>> list("CS101")
['C', 'S', '1', '0', '1']
```

Back to medals

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 ) ]
```

Back to medals

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)
```

Back to medals

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)

def print_totals2():
  for item in medals:
    print(item[0] + ":", sum(item[1:]))
```

Histogram

We want to create a histogram of medals:

Code

```
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]))
```

Result

```
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:
```

Computing prime numbers

KAIST

Sieve of Eratosthenes

Code

```
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]
```