WHAT WE HAVE LEARNED

How to import module

How to create objects
  - Various typed object

Variables, operators and expressions

Functions

Scope of variables

Conditional expression

Loops
How to handle sequences - *lists, strings, tuples*
Here is a table of Olympic medals from the 2010 Vancouver 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>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>4</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Belarus</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Canada</td>
<td>14</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>China</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Croatia</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Czech Republic</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Estonia</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Finland</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>France</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Germany</td>
<td>10</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>Great Britain</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Italy</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Japan</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Kazakhstan</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Korea</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Latvia</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Netherlands</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Norway</td>
<td>9</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Poland</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Russian Federation</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Slovakia</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Slovenia</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Sweden</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Switzerland</td>
<td>6</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>United States</td>
<td>9</td>
<td>15</td>
<td>13</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']
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]
silver=[1, 6, 1, 7, 2, 2, 0, 1, 1, 3, 13, 0, 1, 3, 1, 6, 2, 1, 8, 3, 5, 1, 2, 2, 0, 15]
bronze=[0, 6, 1, 5, 4, 1, 4, 0, 4, 6, 7, 0, 3, 2, 0, 2, 0, 3, 6, 2, 7, 1, 1, 4, 3, 13]
```

A list is an object of type list.

We can access the elements of a list using an integer index.

```python
>>> countries[0]
'Australia'
>>> countries[15]
'Korea'
>>> gold[15]
6
```

Negative indices start at the end of the list:

```python
>>> countries[-1]
'United States'
>>> countries[15:-10]
'Korea'
```
The length of a list is given by `len`

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

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

Lists can contain a mixture of objects of any type

```python
>>> korea = ['Korea', 'KR', 6, 6, 2 ]
>>> korea[1]
'KR'
>>> korea[2]
6
```
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']
```
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
>>> 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.append('D')
>>> list2[1] = 'x'
>>> list1 == list2
True
```
**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
>>> 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]
>>> silver=[1, 6, 1, 7, 2, 2, 0, 1, 1, 3, 13, 0, 1, 3, 1, 6, 2, 1, 8, 3, 5, 1, 2, 2, 0, 15]
>>> broze=[0, 6, 1, 5, 4, 1, 4, 0, 4, 6, 7, 0, 3, 2, 0, 2, 0, 3, 6, 2, 7, 1, 1, 4, 3, 13]
>>> 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)
```
TRAVERSING A LIST

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

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

The range function returns a list:

```python
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> range(10, 15)
[10, 11, 12, 13, 14]

>>> range(10, 15, 2)
[10, 12, 14]
```

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

```python
>>> kk = range(1, 11)

>>> for i in range( len(kk) ):
...     kk[i] = kk[i] ** 2

>>> kk
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```
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)`. 

```
```
SORTING AND REVERSING

We can sort a list using its sort method:

```
>>> ta = ['Albert', 'Jane', 'Brown', 'Barrack', 'John']
>>> ta.sort()
>>> ta
['Albert', 'Barrack', 'Brown', 'Jane', 'John']
```

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')]
```
Slicing creates a new list with elements of the given list:

```
sublist = mylist[m:n]
```

Then `sublist` contains elements $m, m+1, \ldots, n-1$ of `mylist`.

- If $m$ is omitted, the sublist starts with the first element.
- If $n$ is omitted, then the sublist ends with the last element.

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

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

Actually we only care about the top-ten:

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

Unpack immediately
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>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canada</td>
<td>14</td>
<td>7</td>
<td>5</td>
<td>26</td>
</tr>
<tr>
<td>Germany</td>
<td>10</td>
<td>13</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>United States</td>
<td>9</td>
<td>15</td>
<td>13</td>
<td>37</td>
</tr>
<tr>
<td>Norway</td>
<td>9</td>
<td>8</td>
<td>6</td>
<td>23</td>
</tr>
<tr>
<td>Korea</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>Switzerland</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Sweden</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>China</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>Austria</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>Netherlands</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>
Let’s find all countries that have only one kind of medal:

```python
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 concatenation
List objects L have the following methods:

- `L.append(v)` adds object v at the end
- `L.insert(i, v)` inserts element at position i
- `L.pop()` removes and returns last element
- `L.pop(i)` removes and returns element at position i
- `L.remove(v)` removes first element equal to v
- `L.index(v)` returns index of first element equal to v
- `L.count(v)` returns number of elements equal to v
- `L.extend(K)` appends all elements of sequence K to L
- `L.reverse()` reverses the list
- `L.sort()` sorts the list

What is the difference?

L.append(13)
L + [13]

The principle of command-query separation
Lists are a kind of sequence. We already met other kinds of sequences: strings, and tuples:

```python
>>> a = 'ECP'
>>> a[0]
'E'
>>> a[-1]
'P'
>>> a[1:]
'CP'
>>> for i in a:
...    print i,
ECP
>>> for i in a:
...    print i
ECP

>>> t = ('ECP', 'A+', 10)
>>> t[0]
'ECP'
>>> t[0][2]
'P'
>>> t[-1]
10
>>> t[1:]
('A+', 10)
>>> for i in t:
...   print i,
ECP A+ 10
```
Lists and tuples are very similar, but lists are mutable, while tuples (and strings) are immutable:

```python
>>> t[0] = "ECE204"
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)
['ECP', 'A+', 10]
>>> tuple(gold)
(2, 4, 1, 14, 5, 0, 2, 0, 0, ..., 0, 5, 6, 9)
>>> list("ECP")
['E', 'C', 'P']
```
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', 2, 1, 0 ),
           ( 'Austria', 4, 6, 6 ),
           ...
           ( 'United States', 9, 15, 13 ) ]
```

Print total number of medals for each country:

```python
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:])
```
TOP TEN AGAIN

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

```python
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:])
```

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

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

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 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 k in range(len(t)-1, i, -1):
            if t[k] % p == 0:
                t.pop(k)
        i += 1
    return t
Strings are **immutable sequences**

```python
def is_palindrome(s):
    for i in range(len(s) // 2):
        if s[i] != s[len(s) - i - 1]:
            return False
    return True
```

The **in** operator for strings:

```python
>>> "abc" in "01234abcde"
True
>>> "abce" in "01234abcde"
False
```

cf) For tuple and list, **in** tests whether something is equal to an element of the list or tuple
## STRING SPECIAL OPERATORS

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Concatenation - Adds values on either side of the operator</td>
<td>a + b will give HelloPython</td>
</tr>
<tr>
<td>*</td>
<td>Repetition - Creates new strings, concatenating multiple copies of the same string</td>
<td>a*2 will give -HelloHello</td>
</tr>
<tr>
<td>[]</td>
<td>Slice - Gives the character from the given index</td>
<td>a[1] will give e</td>
</tr>
<tr>
<td>[: ]</td>
<td>Range Slice - Gives the characters from the given range</td>
<td>a[1:4] will give ell</td>
</tr>
<tr>
<td>in</td>
<td>Membership - Returns true if a character exists in the given string</td>
<td>H in a will give 1</td>
</tr>
<tr>
<td>not in</td>
<td>Membership - Returns true if a character does not exist in the given string</td>
<td>M not in a will give 1</td>
</tr>
<tr>
<td>r/R</td>
<td>Raw String - Suppress actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter &quot;r,&quot; which precedes the quotation marks. The &quot;r&quot; can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark.</td>
<td>print r'\n' prints \n and print R'\n' prints \n</td>
</tr>
<tr>
<td>%</td>
<td>Format - Performs String formatting</td>
<td>See at next section</td>
</tr>
</tbody>
</table>

\( a = \) Hello  
\( b = \) Python
String methods have many useful methods:
  - `upper()`, `lower()`, and `capitalize()`
  - `isalpha()` and `isdigit()`
  - `startswith(prefix)` and `endswith(suffix)`
  - `find(str)`, `find(str, start)`, and `find(str, start, end)`
    (return -1 if str is not in the string)
  - `replace(str1, str2)`
  - `rstrip()`, `lstrip()` and `strip()` to remove white space on the right, left, or both ends.

String methods for converting between lists and strings:
  - `split()` splits with white space as separator
  - `split(sep)` splits with given separator sep
  - `join(l)` concatenates strings from a list l
  - `repr()` or `str()` converts any value to a string

```
>>> str(39.95)        >>> repr(39.95)
'39.95'               '39.950000000000003'
```
Download blackjack.py and blackjack.docx from icampus

Read the document to understand the rule of ‘BlackJack’

Understand the main part in blackjack.py

Implement the following functions

- create_deck()
- hand_value(hand)
- card_string(card)
- ask_yesno(prompt)
Welcome to Black Jack!

You are dealt a 6 of Spades
Dealer is dealt a hidden card
You are dealt a 2 of Diamonds
Dealer is dealt a 4 of Hearts
Your total is 8
Would you like another card? (y/n) *ne
I beg your pardon!
Would you like another card? (y/n) y
You are dealt an Ace of Diamonds
Your total is 19
Would you like another card? (y/n) n

The dealer’s hidden card was an 8 of Hearts
Dealer is dealt a 10 of Clubs
The dealer’s total is 22

Your total is 19
The dealer’s total is 22
The dealer went over 21! You win!

Play another round? (y/n) n

Initial two drawers
Initial prompt
Report an error if your input is not correct
Type “y” if you want to have another card, otherwise type “n”
When your turn is finished, the dealer draws until the score reached 17.
Show your scores and decide who wins.
The game should go on unless typed “n”