Lecture 24: print revisited, tuples cont.
print
print revisited

We often convert to strings when printing variables:

```python
print("The sum is " + str(sum))
```

This is not always necessary. Definition of built-in function `print`:

```python
print(value, ..., sep=' ', end='
')
```

Prints the values to the console.

Optional keyword arguments:

- `sep`: string inserted between values, default a space.
- `end`: string appended after the last value, default a newline.

Can use this instead:

```python
print("The sum is", sum)
```
print revisited

Examples:

```python
>>> x = 10
>>> y = 20
>>> print(x, y)
10 20
>>> print("x =", x)
x = 10
>>> print("x =", x, "y =", y)
x = 10 y = 20
>>> print("x =", x, " and ", "y =", y)
x = 10 and y = 20

```

Note that the default value for the `sep=` option is providing the space after the "="
print revisited

This works for lists and tuples also:

```python
>>> alist = ['ab', 'cd', 'ef']
>>> print(alist)
['ab', 'cd', 'ef']
>>> t = ('Fundamental Algorithms', 'Knuth', 1968)
>>> print(t)
('Fundamental Algorithms', 'Knuth', 1968)
>>> print(x, y, alist)
10 20 ['ab', 'cd', 'ef']
```

We can use the `sep` option to specify the separator:

```python
>>> print(x, y, alist, sep='--')
10--20--['ab', 'cd', 'ef']
```
help
- **help()**: a function in Python that gives information on built-in functions and methods.
- Looking at documentation is a skill that programmers need to develop.
- You may not understand the documentation completely, but try it.

```python
>>> help(print)
Help on built-in function print in module builtins:

print(...)  
print(value, ..., sep=' ', end='
', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.

>>>
**help():** for built-in methods, specify the type and method name using the dot notation.

```python
>>> help(str.split)
Help on method_descriptor:

split(...)  
S.split(sep=None, maxsplit=-1) -> list of strings
```

Return a list of the words in S, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done. If sep is not specified or is None, any whitespace string is a separator and empty strings are removed from the result.

```python
>>>`
```
Creating Tuples – a note

Syntax for creating a tuple:

\[(value0, value1, ... , valueN)\]

Example:

\[("Tucson", 90, 60)\]

The parenthesis are optional. The comma is the tuple constructor:

```python
>>> t = "Tucson", 90, 60
>>> t
('Tucson', 90, 60)
>>> t[0]
'Tucson'
```
Tuples

This can lead to confusing bugs:

... 
>>> val = 40, 
>>> sum += val

Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    sum += val
TypeError: unsupported operand type(s) for +=: 'int' and 'tuple'

>>> 

If you get an unexpected type error, look for an unexpected comma.
Using tuples

• Items are accessed via subscripting:
  
  \[ t = ("Tucson", 90, 60) \]
  
  \[ x\_\text{coord} = t[1] \]

• You can loop through tuples the same as lists and strings

<table>
<thead>
<tr>
<th>operation</th>
<th>cal</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>len()</td>
<td><code>len((1, 2, 3))</code></td>
<td>3</td>
</tr>
<tr>
<td>+</td>
<td><code>(1, 2, 3) + (4, 5, 6)</code></td>
<td><code>(1, 2, 3, 4, 5, 6)</code></td>
</tr>
<tr>
<td>*</td>
<td><code>('Hi', 1) * 2</code></td>
<td><code>('Hi', 1, 'Hi', 1)</code></td>
</tr>
<tr>
<td>in</td>
<td><code>3 in (1, 2, 3)</code></td>
<td>True</td>
</tr>
<tr>
<td>for</td>
<td>for <code>x</code> in <code>(1,2,3)</code>:</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>print(x)</code></td>
<td>1 2 3</td>
</tr>
<tr>
<td>min()</td>
<td><code>min((1, 3))</code></td>
<td>1</td>
</tr>
<tr>
<td>max()</td>
<td><code>max((1, 3))</code></td>
<td>3</td>
</tr>
</tbody>
</table>
Lists of tuples

• Given the list below:

```python
>>> all_months = [('january', 31), ('february', 28), ('march', 31),
                ('april', 30), ('may', 31), ('june', 30),
                ('july', 31), ('august', 31), ('september', 30),
                ('october', 31), ('november', 30), ('december', 31)]
```

Write code for two different ways to print each tuple in `all_months`:

1) 
2)
Lists of tuples

• Given the list below:

```python
>>> all_months = [('january', 31), ('february', 28), ('march', 31), ('april', 30), ('may', 31), ('june', 30), ('july', 31), ('august', 31), ('september', 30), ('october', 31), ('november', 30), ('december', 31)]
```

1) Write the code to print the days of all the tuples in `all_months`:
Club problem

• Write a program that maintains information about members of a club.
  • The club maintains the name, the birthday month, and a count of attendance for each member.
  • The membership file is kept in "members.txt"
  • The program provides the user with the three options shown below.

Select one of the following options:
  1. Generate a birthday list:
  2. Remove a member:
  3. Update attendance:
Select 1, 2, or 3:
Club problem

• If option 1 is selected, the program prompts the user for a birthday month and then writes the names of all users with that birthday month to a file called "birthdays.txt".

• If option 2 is selected, the program prompts for the name of the member to be removed and removes that member from membership list.

• If option 3 is selected, the program updates the attendance count for all members.

• The file "members.txt" is updated afterwards to reflect the changes made.

Select one of the following options:

1. Generate a birthday file:
2. Remove a member:
3. Update attendance:

Select 1, 2, or 3:
Club problem

Assume that the user input will be correct (no error checking needed). The format of "members.txt" is shown below:

mary october 31
sue april 46
tom march 52
kylie april 24
ben june 45
sally april 22
harry june 48
ann march 44
steve august 55

What data structure best suits this problem?
def main():
    member_list = get_members("members.txt")
    print("Select one of the following options:")
    print("1. Generate a birthday list:")
    print("2. Remove a member:")
    print("3. Update attendance:")
    n = int(input("Select 1, 2, or 3: "))
    if (n == 1):                # generate birthday list file
        month = input("Enter birthday month: ")
        generate_bdays(member_list, month.lower(), "birthdays.txt")
    elif (n == 2):              # remove member
        name = input("Enter name of member to remove: ")
        remove_member(member_list, name.lower())
    else:                       # update attendance of all members
        update_attendance(member_list)
    update("members.txt", member_list)
def get_members(fname):
    f = open(fname)
    lines = f.readlines()

    members = []
    # initialize the list

    for line in lines:
        info = line.split()
        name = info[0]
        bd_month = info[1]
        days_attended = info[2]
        members.append((name, bd_month, days_attended))  # add a tuple of info
    return members
Let's write the code for option 3: update attendance for each member.

```python
def update_attendance(mlist):
```

Club solution
# Increment the attendance count of all members by one.

def update_attendance(mlist):
    for i in range(0, len(mlist)):
        member = mlist[i]  # get the ith member - a tuple
        # replace the ith element with a new tuple
        mlist[i] = (member[0], member[1], int(member[2]) + 1)
    return