Top-Down Programming Example: Rock, Paper, Scissors

Step #1: Goal and General Algorithm Idea

Goal: write a game to play "rock, paper, scissors"
The user chooses one of these, the computer chooses the other

- If the pair is "rock, paper", the paper wins
- If the pair is "scissors, paper", the scissors wins
- If the pair is "scissors, rock", the rock wins

Specification: user enters selection of rock, paper, scissors
Program prints computer's selection, who wins
At end, computer prints number of games human won and it won
High-level design:

```
initialize score
loop
    ask user for choice
    if quit, exit loop
    computer selects one
    select winner and increment win count
endloop
print number of games user won, computer won, ties
```

Step #2: Data Representation and Program Structure

Part #1: Data
Represent the rock, paper, scissors using strings: "rock", "paper", "scissors" (sequence *things*)
Represent commands as strings as above, plus "quit" (sequence *cmdlist*)
Store the scores in a dictionary with keys "user", "computer", "tie" and integer values (initially set to 0)
Part #2: Functions

- get user input – *getuser*()
- get computer choice – *getcomp*()
- determine winner – *whowins*()

Part #3: Refine algorithm

```
while True:
    userchoice = getuser();
    if (userchoice == quit):
        break
    compchoice = getcomp();
    winner = whowins(userchoice, compchoice)
    score[winner] += 1
print You won, score["user"], game(s), the computer won, score["computer"], game(s)
print and you tied, score["tie"], game(s)
```

Step #3: Figure out who wins

Represent ($object_1$, $object_2$) where $object_1$ beats $object_2$ as list of tuples, *winlist*. To see if user won, see if the (*user-chosen object*, *computer-chosen object*) tuple is in that list.

This leads to [*rps-prog1.py*]:

```
def whowins(user, comp):
    if user == comp:
        win = "tie"
    elif (user, comp) in winlist:
        win = "user"
    else:
        win = "computer"
    return win
```

Step #4: Get computer choice

Given the three objects in the sequence *things*, choose randomly.

This leads to [*rps-prog2.py*]:

```
def getcomp():
    pick = random.choice(things)
    print("Computer picks", pick)
    return pick
```

Step #5: Get user input

Loop until you get a valid input. If the user types an end of file (control-d) or an interrupt (control-c), act as though the user typed "quit"; report any other exceptions and then act as though the user typed "quit".

This leads to [*rps-prog3.py*]:

```
def getuser():
    while True:
        try:
            n = input("Human: enter rock, paper, scissors, quit: ")
        except (EOFError, KeyboardInterrupt):
            n = "quit"
            break
        except Exception as msg:
            print("Unknown exception:", msg, "-- quitting")
            n = "quit"
            break
        *** check input ***
    return n
```

To check input, we need to be sure it's a valid command, so see if it's in *cmdlist*:

```
        if n not in cmdlist:
            print("Bad input; try again")
        else:
            break
```

Put these together to get the user input routine.