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
We can now put this into Python (see *rps-1.py*):

```
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", end=" ")
print(score["computer"], "game(s), and you two tied", score["tie"], "game(s)")
```

Step #3: Figure out who wins

Represent ($object_1$, $object_2$) where $object_1$ beats $object_2$ as a list of pairs called *winlist*. To see if user won, check if the *user-chosen object* is the same as the *computer-chosen object*; if so, there is no winner. Otherwise, check whether (*user-chosen object*, *computer-chosen object*) pair is in *winlist*; if so, the user wins. Otherwise the user loses.
This leads to *rps-2.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-3.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 (usually control-D) or an interrupt (usually 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-4.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.

Step #6: Make it human-friendly

The program now works correctly, but it's rather unfriendly— the "game(s)" should be "game" or "games" as appropriate, and it should tell the user who wins each round. So we need to add something to the `while True` loop in the main routine, and change the `print` statements at the end.

Telling the user who wins is straightforward. Simply put in an `if` statement at the end of the loop. One tricky point is that there are actually four conditions: `winner` can take on three known values ("user", "computer", and "tie"), and any other unknown value. It should never do the latter, but just in case, we program defensively and put a special case in to catch that. The resulting code is:

```
if winner == "user":
        print("You win")
elif winner == "computer":
        print("Computer wins")
elif winner == "tie":
        print("Tie")
else:
        print( "*** INTERNAL ERROR *** winner is", winner)
        break
```

Next, the program should distinguish between 1 "game" and any other number of "games" (note you say "0 games" in English). Again, we use an `if` statement to handle it. Both the computer's number of games, the user's number of game, and the number of tie games have to be handled.

```
print("You won", end='')
if score["user"] == 1:
        print("1 game, the computer won",end=' ')
else:
        print(score["user"], "games, the computer won", end=' ')
    if score["computer"] == 1:
        print("1 game, and you two tied", end=' ')
    else:
        print(score["computer"], "games, and you two tied", end=' ')
    if score["tie"] == 1:
        print("1 game.")
    else:
        print(score["tie"], "games.")
```

The resulting program is *rps-5.py*.