NEA sample solution
Task 3 – Card trick

GCSE Computer Science 8520
NEA (8520/CA/CB/CC/CD/CE)

Introduction

The attached NEA sample scenario solution is provided to give teachers an indication of the type of solution that students could complete in response to the specimen material NEA scenario: Task 3 – Card trick for the new GCSE Computer Science (8520) specification. This specification is for first teaching from September 2016.

This sample solution should only be used to enable teachers to commence planning work for the NEA, (the live NEA scenario will be available for the first time from September 2017). As a result teachers should use this only as a guide to the forthcoming live scenarios. This solution is not a 'real' solution and is provided as an example only. It cannot be used to accurately determine standards in the future.

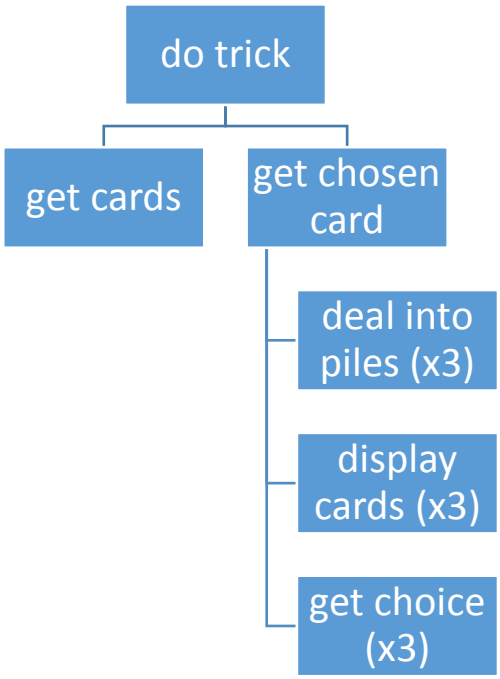
At our preparing to teach events, spring 2016, the sample scenarios and solutions will be considered and discussed. After these events, appropriate commentaries will be provided, by the senior examining team, to enable teachers to understand the appropriate level achieved by this solution.

Teacher Standardisation events will be used to prepare teachers for the first NEA assessment, which will be available in centres from September 2017. At these meetings teachers will be made aware of the standard.

AQA NEA Card Trick

Designing the Solution

The top-level design is shown here:



The interface for this design is:

- Subroutine **do_trick**
  - **inputs**: none
  - **output**: none
  - **purpose**: top-level subroutine, contains everything to complete the card trick
- Subroutine **get_cards**
  - **inputs**: none
  - **output**: cards (list of strings)
  - **purpose**: creates a list of strings representing 21 playing cards and then shuffles these cards
- Subroutine **get_chosen_card**
  - **inputs**: cards (list of strings)
  - **output**: chosen card (string)
  - **purpose**: performs the three deals of the cards, for each one getting the user choice.
- Subroutine **deal_into_piles**
  - **inputs**: cards (list of strings)
  - **output**: list of three piles of cards (list of list of strings)
  - **purpose**: takes the whole deck of cards and deals it into the three piles working left to right and top to bottom
- Subroutine **display_cards**
  - **inputs**: three piles of cards (three lists of strings)
  - **output**: none
  - **purpose**: outputs the cards so that they appear to be in three distinct piles

- Subroutine **get_choice**
  - **inputs**: the number of options the user can choose (integer)
  - **output**: the user's choice (integer)
  - **purpose**: gets the user choice of 1, 2 or 3 validating that the user has entered an integer in this range

The subroutines will do the following:

do_trick()

1. cards ← get_cards
2. card ← get_chosen_card(cards)
3. output card

get_cards()

1. cards ← ['H A', 'H 2', 'H 3', etc for 21 different playing cards]
2. shuffle cards
3. RETURN cards

get_chosen_card(cards)

1. REPEAT 3 times…
   a. [pile1, pile2, pile3] ← deal_into_piles(cards)
   b. display_cards(pile1, pile2, pile3)
   c. pile_choice ← get_choice(3)
   d. if choice = 1 then cards ← pile2 + pile1 + pile3
   e. if choice = 2 then cards ← pile1 + pile2 + pile3
   f. if choice = 3 then cards ← pile1 + pile3 + pile2
2. RETURN the middle card in cards

deal_into_piles(cards)

1. create three empty lists called pile1, pile2 and pile3
2. FOR i ← 0 to length(cards)…
   a. if i MOD 3 = 0 then append cards[i] to pile1
   b. if i MOD 3 = 1 then append cards[i] to pile2
   c. if i MOD 3 = 2 then append cards[i] to pile3
3. RETURN [pile1, pile2, pile3]

display_cards(pile1, pile2, pile3)

1. FOR i ← 0 to length(any of the piles)…
   a. print pile1[i] SPACE pile2[i] SPACE pile3[i]

get_choice(options)

1. WHILE true…
   a. choice ← INPUT
   b. if choice is an integer and >=1 and <=options then RETURN choice

Creating the Solution

```
import random

# main subroutine of the program
def do_trick():
    # get 21 random cards
    cards = get_cards()
    # find the chosen card
    card = get_chosen_card(cards)
    print("your card is...")
    # display the chosen card
    print(card)

# subroutine that takes the 21 shuffled cards as input and
# returns the chosen card
def get_chosen_card(cards):
    print("choose a card and remember it...")
    # the cards are dealt three times and so a for loop
    # performs the dealing and selection 3 times
    for x in range(3):
        pile1, pile2, pile3 = deal_into_piles(cards)
        print("which pile is your card in...")
        # call to the display_cards subroutine with the three piles as input
        display_cards(pile1, pile2, pile3)
        # the user choice is validated using the get_choice subroutine
        pile_choice = get_choice(3)
        # depending on the user choice the piles are concatenated in different
orders
        if pile_choice == 1:
            cards = pile2 + pile1 + pile3
        elif pile_choice == 2:
            cards = pile1 + pile2 + pile3
        else:
            cards = pile1 + pile3 + pile2
    # the chosen card is at the midpoint so integer division is used
    card_index = len(cards) // 2
    # the chosen card is returned
    return cards[card_index]

# subroutine to shuffle and return a deck of 21 cards
def get_cards():
    # the cards are stored as an array of strings
    cards = ["H A", "H 2", "H 3", "H 4", "H 5", "H 6", "H 7",
             "S A", "S 2", "S 3", "S 4", "S 5", "S 6", "S 7",
             "D A", "D 2", "D 3", "D 4", "D 5", "D 6", "D 7"]
    # the deck is shuffled and then returned
    random.shuffle(cards)
    return cards

# subroutine to deal cards into three piles
def deal_into_piles(cards):
    # three empty piles are intiated for every iteration
    pile1 = []
    pile2 = []
    pile3 = []
    # iterates over the whole deck using the indices of the
    # the elements (cards)
    for index in range(len(cards)):
        # if the index MOD 3 is 0 then the card is appended to the first pile
        if index%3 == 0:
```

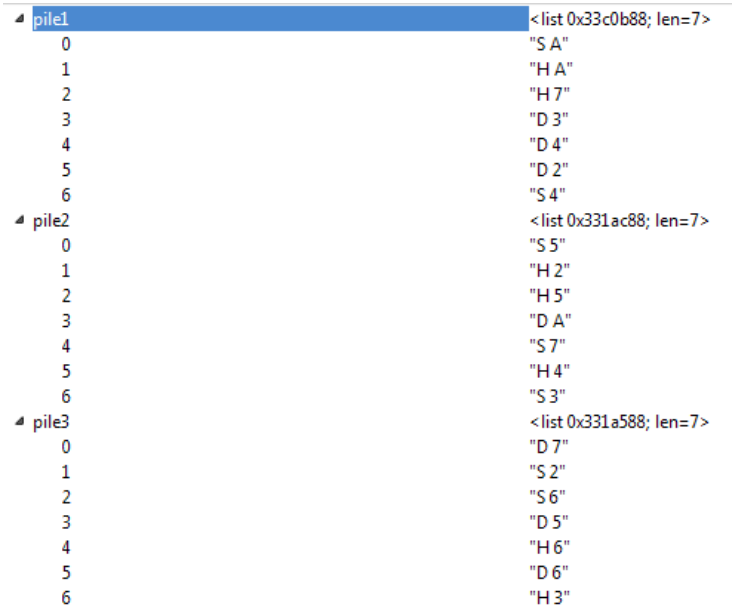
```
            pile1.append(cards[index])
        # if the index MOD 3 is 1 then the card is appended to the second pile
        elif index%3 == 1:
            pile2.append(cards[index])
        # otherwise the card is appended to the third pile
        else:
            pile3.append(cards[index])
    return [pile1, pile2, pile3]

# subroutine to display three piles of cards
def display_cards(pile1, pile2, pile3):
    print("Pile1\t\tPile2\t\tPile3")
    # x is an index that is used to index the particular card
    # in piles 1, 2 and 3 that should be displayed - this makes
    # it easier for the user to 'see' the three piles
    for x in range(len(pile1)):
        print(pile1[x], end="\t\t")
        print(pile2[x], end="\t\t")
        print(pile3[x])
    print()

# validates user choice
def get_choice(options):
    # only returns when the user enters an integer that
    # is between 1 and the options given as input to the subroutine,
    # until then it loops
    while True:
        print("choose option 1 to", options)
        try:
            choice = int(input(">>>"))
            # range check performed on the integer
            if choice >= 1 and choice <= options:
                # if it passes the type and range check then it is returned
                return choice
            else:
                print("invalid range")
        # exception caught if the user does not enter a value that
        # can be converted to an integer
        except:
            print("not a number")

# calls the main subroutine
do_trick()
```

Testing the Solution

Test	Purpose	Input/Action	Expected Result	Actual Result
1	Check that the cards display correctly	Run the program	Inspect the program (using watches) to check that the three piles are displayed correctly	The internal lists of pile1, pile2 and pile3 are displayed correctly
2	The list of 21 cards is randomised at the start of the game	Run the program three times	Check that the order of the cards is different	Cards are obviously in a different order each time
3	Check that the program deals the cards into three piles	Run the program	Inspect the program (using watches) to compare the initial list of cards to the three piles and check that the first, second and final third match the three piles	The initial list of cards matches the piles
4.1	Check user input (correct range)	User enters 4 (erroneous, boundary)	Not accepted and user prompted again	Not accepted and prompted again
		User enters 0 (erroneous, boundary)	Not accepted and user prompted again	Not accepted and prompted again
		User enters 1 (correct, boundary)	Accepted	Accepted
		User enters 3 (correct, boundary)	Accepted	Accepted
4.2	Check user input (type check)	User enters ‘a’	Not accepted and user prompted again	Not accepted and prompted again
4.3	Check user input (presence check)	User enters nothing	Not accepted and user prompted again	Not accepted and prompted again
5	Cards rearranged correctly according to user choice of pile	User enters 1	Check that the cards in pile 1 are in the middle third of the new list of cards	They are
		User enters 2	Check that the cards in pile 1 are in the middle third of the new list of cards	They are
		User enters 3	Check that the cards in pile 1 are in the middle third of the new list of cards	They are

Test	Purpose	Input/Action	Expected Result	Actual Result
6	Check overall correctness	Run the program three times, each time selecting the Ace of Spades	For each game, the program will ask for user input three times and then output the chosen card is the Ace of Spades	Every time works correctly

Test Number	Evidence

1	This is the values of the three piles:
	 
	And this is the matching output to the user:
	`which pile is your card in...`
	`Pile1 Pile2 Pile3`
	`S A S 5 D 7`
	`H A H 2 S 2`
	`H 7 H 5 S 6`
	`D 3 D A D 5`
	`D 4 S 7 H 6`
	`D 2 H 4 D 6`
	`S 4 S 3 H 3`

2 All three runs show the cards in different order.

First run:

cards	<list 0x2e5a588; len=21>
0	"D A"
1	"H 4"
2	"S A"
3	"S 3"
4	"S 7"
5	"H 3"
6	"D 2"
7	"D 6"
8	"S 5"
9	"D 4"
10	"D 5"
11	"H 5"
12	"D 7"
13	"H 6"
14	"H 2"
15	"D 3"
16	"H A"
17	"S 6"
18	"S 4"
19	"H 7"
20	"S 2"

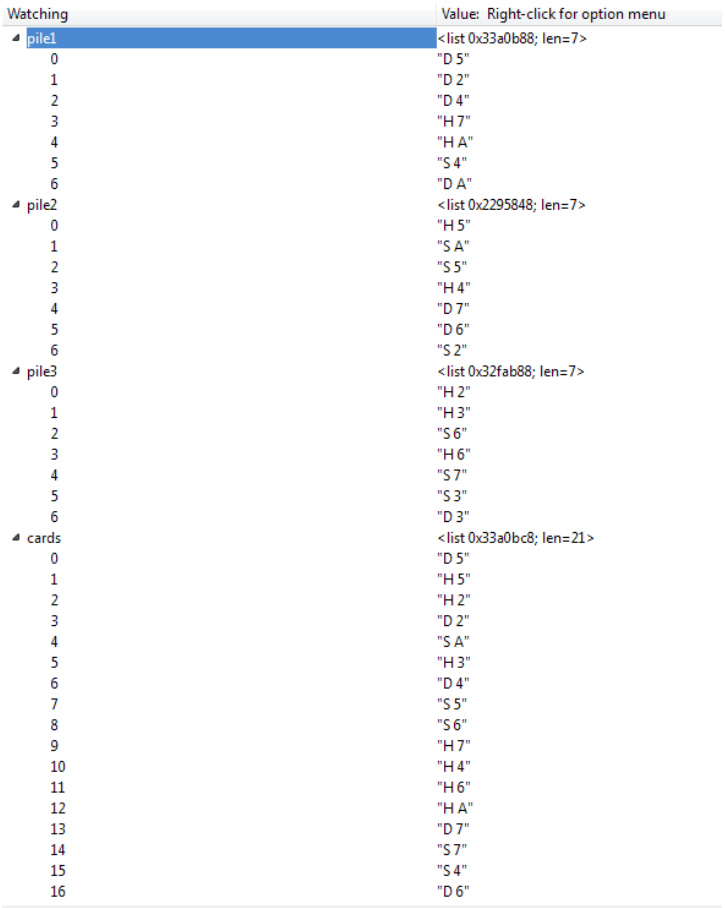
Second run:

cards	<list 0x3370bc8; len=21>
0	"H 6"
1	"S 2"
2	"S A"
3	"H A"
4	"H 7"
5	"D 3"
6	"S 6"
7	"D A"
8	"H 2"
9	"H 3"
10	"H 4"
11	"S 7"
12	"H 5"
13	"D 6"
14	"D 4"
15	"S 4"
16	"D 2"
17	"D 5"
18	"D 7"
19	"S 5"
20	"S 3"

Third run:

cards	<list 0x3308748; len=21>
0	"S A"
1	"H 3"
2	"H 4"
3	"D 4"
4	"S 6"
5	"H 6"
6	"S 2"
7	"S 7"
8	"D A"
9	"S 3"
10	"D 2"
11	"D 3"
12	"D 6"
13	"H A"
14	"S 5"
15	"S 4"
16	"H 7"
17	"H 5"
18	"H 2"
19	"D 7"
20	"D 5"

3



Watching	Value: Right-click for option menu
pile1	<list 0x33a0b88; len=7>
0	"D 5"
1	"D 2"
2	"D 4"
3	"H 7"
4	"H A"
5	"S 4"
6	"D A"
pile2	<list 0x2295848; len=7>
0	"H 5"
1	"S A"
2	"S 5"
3	"H 4"
4	"D 7"
5	"D 6"
6	"S 2"
pile3	<list 0x32fab88; len=7>
0	"H 2"
1	"H 3"
2	"S 6"
3	"H 6"
4	"S 7"
5	"S 3"
6	"D 3"
cards	<list 0x33a0bc8; len=21>
0	"D 5"
1	"H 5"
2	"H 2"
3	"D 2"
4	"S A"
5	"H 3"
6	"D 4"
7	"S 5"
8	"S 6"
9	"H 7"
10	"H 4"
11	"H 6"
12	"H A"
13	"D 7"
14	"S 7"
15	"S 4"
16	"D 6"

It's clear that the first element of cards is the first element of pile1, the second element of cards is the first element of pile2, the third element of cards is the first element of pile3 and so on.

4

All of the erroneous data is shown here (each one is not accepted):

```
choose option 1 to 3
>>>4
invalid range
choose option 1 to 3
>>>0
invalid range
choose option 1 to 3
>>>a
not a number
choose option 1 to 3
>>>
not a number
choose option 1 to 3
>>>
```

These are the two boundary cases that are accepted:

```
choose option 1 to 3
>>>1
which pile is your card in...
Pile1           Pile2           Pile3
H 2             D 2             D 4
D 7             H 7             S 4
S 3             H A             S 5
S 6             D 5             D 3
H 5             D A             H 4
S 2             S 7             H 3
H 6             S A             D 6

choose option 1 to 3
>>>2
which pile is your card in...
Pile1           Pile2           Pile3
H 2             D 7             S 3
S 6             H 5             S 2
H 6             D 2             H 7
H A             D 5             D A
S 7             S A             D 4
S 4             S 5             D 3
H 4             H 3             D 6
```

5 By looking carefully at the rearrangement of the cards it is clear that the rearrangement and dealing works.

When the user enters 1:

```
choose a card and remember it...
which pile is your card in...
Pile1           Pile2           Pile3
H A             D 5             S 6
H 3             D 7             H 5
S 7             S 3             H 4
S 2             H 6             D 3
H 7             H 2             S 5
S 4             D A             D 4
D 2             D 6             S A

choose option 1 to 3
>>>1
which pile is your card in...
Pile1           Pile2           Pile3
D 5             D 7             S 3
H 6             H 2             D A
D 6             H A             H 3
S 7             S 2             H 7
S 4             D 2             S 6
H 5             H 4             D 3
S 5             D 4             S A
```

When the user enters 2:

```
choose a card and remember it...
which pile is your card in...
Pile1           Pile2           Pile3
D 7             S 6             S 7
H 7             S 2             S 3
H 6             D A             H A
H 5             H 2             D 6
D 4             D 5             S A
S 4             D 3             H 4
D 2             H 3             S 5

choose option 1 to 3
>>>2
which pile is your card in...
Pile1           Pile2           Pile3
D 7             H 7             H 6
H 5             D 4             S 4
D 2             S 6             S 2
D A             H 2             D 5
D 3             H 3             S 7
S 3             H A             D 6
S A             H 4             S 5
```

When the user enters 3:

```
choose a card and remember it...
which pile is your card in...
Pile1           Pile2           Pile3
H 5             H 2             D 6
H 7             H 3             D 7
S 7             D 4             D A
S 5             S 2             D 2
S 3             H A             S 6
D 3             H 6             H 4
S A             S 4             D 5

choose option 1 to 3
>>>3
which pile is your card in...
Pile1           Pile2           Pile3
H 5             H 7             S 7
S 5             S 3             D 3
S A             D 6             D 7
D A             D 2             S 6
H 4             D 5             H 2
H 3             D 4             S 2
H A             H 6             S 4
```

6 These three final outputs from the message are when the user enters the pile numbers for the Ace of Spades. All three output the correct card.

Trick 1:

```
choose option 1 to 3
>>>2
your card is...
S A
```

Trick 2:

```
choose option 1 to 3
>>>1
your card is...
S A
```

Trick 3:

```
choose option 1 to 3
>>>2
your card is...
S A
```

Potential Enhancements and Refinements

The program displays 21 shuffled cards at the start of the program. This uses the in-built shuffle function from the random module in Python. I could have implemented a randomising algorithm myself but it is highly unlikely it will be more effective or efficient than the built-in version.

I use modulo arithmetic to take the list of 21 cards and divide them equally into three piles. This is an efficient solution as it only requires me to iterate over the list of cards once. This is a much more elegant solution than creating three piles from, for example, cards[0], cards[3], cards[6], etc.

I have created a subroutine that validates the user input. This loops until the user has entered an integer that is either 1, 2 or 3. I use try and except to catch if a user has entered a non-integer value (a run-time error occurs when the input string is converted to an integer unless that string can be obviously converted).

I use list concatenation to produce a new list comprising the three different piles. I use IF, ELIF and ELSE here as there are only three possible branches that can be taken.

My program repeats the core functions three times before it uses integer division to find the mid-point of the list of cards and displays that card to the user.

I could enhance my program by asking the user how many cards they would like to use. This needs to be an odd numbered multiple of 3 (e.g. 3, 9, 15, 21, 27, 33, 39, 45 or 51). I would need to change the number of iterations for some of these from 3 to 1, 2 or 4. I would also have to change the user choice to reflect this (my subroutine is already set up to do this).