# Worksheet 3 Stacks Answers

**Task 1 Crushing cars**

1. (a) Complete the following to show the operations implemented on a collection of burnt-out cars. The stack can hold a maximum of 6 items.

 **Cars:** Mondeo, Golf, Fiesta, Punto, Civic, Porsche

 Representation of the stack drawn both horizontally and vertically are shown. Show the state of the stack after each push and pop operation in both representations, and in the first table, show any results returned.

|  |  |  |
| --- | --- | --- |
|  | **Stack** | **Result returned** |
| carStack 🡨 Stack() | [] |  |
| carStack.push (Mondeo) | [Mondeo] |  |
| carStack.push (Golf) | [Mondeo, Golf] |  |
| carStack.isEmpty() |  | False |
| carStack.push(Fiesta) | [Mondeo, Golf, Fiesta] |  |
| carStack.push(Punto) | [Mondeo, Golf, Fiesta, Punto] |  |
| carStack.pop() | [Mondeo, Golf, Fiesta] | Punto |
| carStack.push(Civic) | [Mondeo, Golf, Fiesta, Civic] |  |
| carStack.push(Porsche) | [Mondeo, Golf, Fiesta, Civic, Porsche] |  |
| carStack.isFull()  |  | False |
| carStack.pop() | [Mondeo, Golf, Fiesta,Civic] | Porsche |
| carStack.pop() | [Mondeo, Golf, Fiesta] | Fiesta |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | Punto |
|  |  |  |  |  |  | Fiesta |  | Fiesta |
|  |  |  |  | Golf |  | Golf |  | Golf |
|  |  | Mondeo |  | Mondeo |  | Mondeo |  | Mondeo |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  | Porsche |  |  |  |  |
|  |  | Civic |  | Civic |  | Civic |  |  |
| Fiesta |  | Fiesta |  | Fiesta |  | Fiesta |  | Fiesta |
| Golf |  | Golf |  | Golf |  | Golf |  | Golf |
| Mondeo |  | Mondeo |  | Mondeo |  | Mondeo |  | Mondeo |

2. Complete the pseudocode below for a program which uses a stack to test an input string to determine whether it is a palindrome (the same backwards and forwards, like “peep”)

 Assume that a class **Stack** implements the operations in the table in question 1.

OUTPUT “Please enter a word or phrase to be tested”)

INPUT myString

list1 🡨 list(myString) #convert myString to a list of characters

numChars 🡨 len(list1)

s 🡨 Stack()

#push each character onto the stack

FOR char in list1

 s.push(char)

ENDFOR

list2 = [] #create an empty list

#pop each character off the stack into a second list

FOR char = 0 to numChars - 1

 list2.append(s.pop())

ENDFOR

#compare the two lists, one is the reverse of the other

IF list1 == list2 THEN

 print("This is a palindrome")

ELSE

 print("This is not a palindrome")

ENDIF

(See Python program palindrome.py in folder Topic3 Python programs

**Task 2 Subroutine snake**

3. (a) Fill in the return addresses to show the state of the stack created during the execution of the progam beginning at **main()**. Each {stack frame} is indicated with curly brackets and contains parameters, local variables and return addresses.

|  |  |
| --- | --- |
| 78 | SUB subA(p1) |
| 79 |  subB (p1, 4) |
| 80 |  … |
| 97 | ENDSUB |
|  |  |
| 99 | SUB subB(p10, y) |
| 100 |  x 🡨 12 |
| 101 |  subC (p10, x) |
| 102 |  … |
| 110 | ENDSUB |
|  |  |
| 144 | SUB subC(p3, p4) |
| 145 |  … |
| 146 |  … |
| 151 | ENDSUB |
|  |  |
| 222 | main() |
| 223 |  subA(7) |
| 224 |  … |

|  |  |
| --- | --- |
| **Line** | **Stack** |
| 223 | {call subA: parameter = 7, return address = 224 } |
| 78 |  |
| 79 | {call subA: p1 =7, return address = 224 } { call subB: p10 = 7, y =4, return address = 80 } |
| 99 |  |
| 101 | {call subA: p1 =7, return address = 224 } {call subB: p10 = 7, y =4, x = 12, return address = 80 }{call subC: p3 = 7, p4 = 12, return address = 102 } |
| 144 |  |
| 145 |  |
| 151 | (pop the stack, go to the return address) |
| 102 | {call subA: 7, return address = 224 } {call subB: 7, 4, 12, return address = 80 } |
| 110 | (pop the stack, go to the return address) |
| 80 | {call subA: p1 =7, return address = 224 } |
| 97 | (pop the stack, go to the return address) |
| 224 | … |