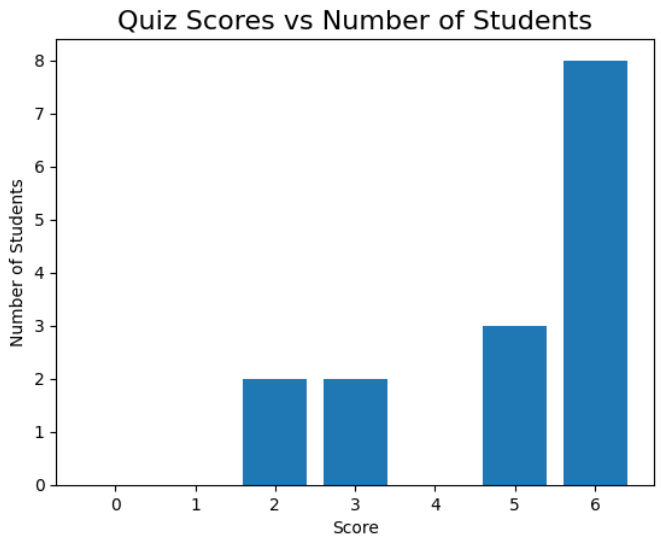
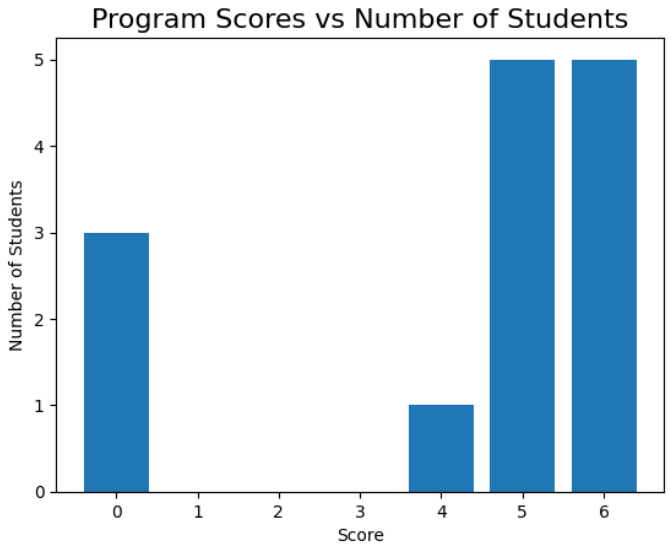
Test 1, observations

- Great participation, nearly everyone attempted!
- Overall students did better in quiz
  - 8x full marks in quiz vs 5x full marks in programming
- 4 students got a zero in programming (with 1 not attempted)





Test 2 , analysis

- The quizzes
  - seem to be easier for students to master
  - Success cause everyone attempted, next we figure out how to raise scores there
    - Suggestions ?
- Programming
  - It was amazing attempt
  - The codes generally worked
  - But 4/15 students had zeroes.
  - Suggestions ?
- Goal :We need raise scores for everyone and move some in attempted stage
  - Note
    - Compete with yourself , Do better than your last attempt
    - Ask for help
      - on what you don’t know, to learn it.
      - And even on what you know , i.e. to master it

Topics for Test 2

- Mathematical expressions
  - Representations (infix, prefix, postfix)
  - Evaluation or expression
- Discrete Mathematics
  - Logical Operators (AND , OR , NOT, SHIFT,…)
  - Boolean logic
- Programming
  - char, string ,arrays
  - Loops
  - Conditionals
    - If/else
    - switch/case

Infix expressions

- Encounter them in grade maths. e.g.
  - (11+14)/(9 - 3) + 2
  - 3+7 / (4 * 5 - 6)
  - - 2 + 8 / 2
- Can also be written using variables
  - (a + b) / (c – d) + e
  - x + y / (g * h – k)
- Evaluated using
  - PEMDAS / BODMAS

Expression : anatomy

- Expression is made of
  - Operators (+ , - , * , /)
  - Operands (numbers, variables)
- Unary operators : - a
  - <operator> <operand>
- Binary operators (infix) : a + b
  - < operand> <operator> <operand>

Prefix/Postfix expressions

- Benefits
  - Remove ambiguity(i.e. can forget PEMDAS, BODMAS)
  - Computer friendly
  - Works with both unary/binary
  - Faster evaluation
- Expression styles
  - Infix : operator in **middle** of operands
    - (5+6)*3
    - < ộřêsắŋđ> <*operator*> <ộřêsắŋđ>
  - Prefix : operator **before** operands
    - * + 5 6 3
    - <*operator*> <ộřêsắŋđ> < ộřêsắŋđ>
  - Postfix : operator **after** operands
    - 5 6 + 3 *
    - <ộřêsắŋđ> < ộřêsắŋđ> <*operator*>

Evaluating expression,
*but first **Stacks** and **Queues***

- What is a **Stack** ?
  - This is a sequential structure of **items**
  - That are **pushed at one end**
  - **Pulled** via the **same end**
  - **LIFO** : Last in First Out
  - Examples
    - Stack of plates
    - Your turned in paper assignments
- What is a **Queue**?
  - Another sequential structure of **items**
  - that are **pushed at one end**
  - **Pulled** via the **opposite end**
  - **FIFO** : First In First Out
  - Examples
    - Queue for buying tickets
    - Traffic in one way single lane

Evaluating **prefix** expression

- Infix : (5+6)*3
  - * + 5 6 3
  - - + 2 * 3 4 / 16 ^ 2 3
- Algorithm (harder way)
  - Scan from right to left
  - Anytime you find an operator
    - Evaluate the operation using the previous 2 operands
    - Replace the <*operator*> <ộřêsắŋđ> < ộřêsắŋđ> with sêṣụľʧ in the expression
  - Repeat the steps till only 1 operand is left

Evaluating postfix expression

- Evaluate
  - 2 3 1 * + 9 –
  - 5 3 + 6 2 / * 3 5 * +
- *Algorithm to evaluate (use **stack**)*
  1. Push the operands in a stack
  2. When you encounter a operator
     - Pop 2 operands
     - Perform the operation on operands
     - Push the result in stack
  - Repeat from 1 until expression is parsed
  - The last item in stack is the answer
    - There would be only 1 item left when done correctly
- * prefix expr can also be evaluated in this way (*just in **reverse***)

Evaluate

- Prefix
  - - * 5 + - 4 2 2 / 6 3
  - - * + 3 5 7 + / 4 2 1
  - - + 10 * 2 3 + 4 / 5 5
- Postfix
  - 1 2 + 3 4 + * 5 6 - / 7 +
  - 8 2 / 3 4 + * 5 1 + 2 / -
  - 9 8 4 2 1 ^ * / - 3 +

Answers

- Prefix
  - - * 5 + - 4 2 2 / 6 3
    - Ans : **18**
  - - * + 3 5 7 + / 4 2 1
    - Ans : **53**
  - - + 10 * 2 3 + 4 / 5 5
    - Ans : **11**
- Postfix
  - 1 2 + 3 4 + * 5 6 - / 7 +
    - Ans : **-14**
  - 8 2 / 3 4 + * 5 1 + 2 / -
    - Ans : **25**
  - 9 8 4 2 1 ^ * / - 3 +
    - Ans : **11**

Practice: Programming

- Write a function that takes a string and checks if it is a palindrome
  - Returns true if palindrome is found else false
  - Hints: string indexing, loops, if , comparing characters
- Count the number of vowels, consonants in a sentence.
  - Vowels : a,e,i,o,u
  - Consonant : everything else other than vowels
  - Ignore : spaces (‘ ’,) comma(,) dash(-),semicolon(;),colon(:)
- Find the most frequent word in a sentence
  - If more than 1 word has same frequency return the lexicographically smaller one.