hrs	/ /20		Exams Office Use Only

University of the Witwatersrand, Johannesburg

Course or topic No(s)	ELEN4010
Course or topic name(s) Paper Number & title	Software Development III
Examination/Test* to be held during month(s) of (*delete as applicable)	June/July 2024
Year of Study (Art & Sciences leave blank)	Fourth
Degrees/Diplomas for which this course is prescribed (BSc (Eng) should indicate which branch)	BSc(Eng)(Elec)(Info)
Faculty/ies presenting candidates	Engineering and the Built Environment
Internal examiners and telephone number(s)	SP Levitt x77209
External examiner(s)	R Cassim
Special materials required (graph/music/drawing paper) maps, diagrams, tables, computer cards, etc)	Computer card for multiple-choice questions

Time allowance

Course Nos	ELEN4010	Hours	2

Instructions to candidates
(Examiners may wish to use this space to indicate, inter alia, the contribution made by this examination or test towards the year mark, if appropriate)

1. Answer all questions
2. Available marks: 77 - Full marks: 70
3. Closed-book exam
4. Basic scientific calculator permitted

Internal Examiners or Heads of School are requested to sign the declaration overleaf

1. As the Internal Examiner/Head of School, I certify that this question paper is in final form, as approved by the External Examiner, and is ready for reproduction.

2. As the Internal Examiner/Head of School, I certify that this question paper is in final form and is ready for reproduction.

(1. is applicable to formal examinations as approved by an external examiner, while 2. is applicable to formal tests not requiring approval by an external examiner—Delete whichever is not applicable)

Name:_____ Signature:_____

(THIS PAGE NOT FOR REPRODUCTION)

Question 1 [Total Marks: 25]

For each of the multiple choice questions below, there may be *more than one correct answer*. Mark the *correct answers* on the multiple choice card that is provided. Each question counts for five marks.

A *negative marking scheme* is used so any incorrect answers which are selected for a particular question will lower your mark for that question. Note, however, that you cannot score less than zero for a question.

1.1 Which of the following statements are *true*?

a) JavaScript is derived from the Java programming language.

b) A JavaScript object is not a primitive JavaScript type.

c) StandardJS provides the specification for JavaScript language.

d) JavaScript statements must end with a semi-colon.

e) The `Number` type in JavaScript is equivalent to the `int` type in C++.

1.2 Which of the following statements are *true*?

a) Any company that publishes an API becomes part of the software supply chain.

b) The purpose of hosting an internal package registry is that the packages can be vetted for malicious code.

c) Simply knowing the name of an internal company package used to be enough to initiate a supply chain attack.

d) One way to fully guarantee that a Node web application is secure and free from malicious code is to avoid using the NPM registry entirely.

e) Exfiltration with regard to software security refers to the penetration of an organisation by an attacker.

1.3 If you want to create a local copy of a remote repository on your computer, you should use the following Git client command:

a) git fork

b) git init

c) git clone

d) git pull

e) None of the above

1.4 Which of the following statements are *true*?

a) A linter compiles source code and reports any errors it finds.

b) It is better to run a linter locally rather than only as part of the CI server build pipeline.

c) A linter is capable of automatically modifying code so that it follows a certain style.

d) Linters cannot improve code quality.

e) A linter performs static code analysis.

1.5 Which of the following are *correct* with regards to a user story?

a) It should describe one thing the software needs to do for the customer.

b) It should be short.

c) It should mention specific technologies.

d) It should be written using language the customer understands.

e) It should propose a solution to a customer need.

Question 2 [Total Marks: 20]

The Independent Electoral Commission publishes election results as a spreadsheet. The columns are: *Province*, *Ward name*, *Ward number*, *Party*, *Provincial party leader*, *Number of votes* (number of votes the party got in that ward). Within each province the ward name is unique. The ward number is unique across the whole country.

a) Give the SQL code to create a table that would store this data in the format given. (4 marks)

b) Give the SQL code to find how many votes the Pastafarian Party won in the Linguini ward in Gauteng. (3 marks)

c) Give the SQL code to find how many votes the Pastafarian Party won across the whole country. (1 mark)

d) Give the SQL code to find how many wards the Pastafarian Party received more votes than the National Pasta Front. (3 marks)

e) What normal form is the table in? Explain. (1 mark)

f) Give the SQL declarations necessary for creating the tables which would store this data in third normal form. (6 marks)

g) Give one advantage and one disadvantage of the normalised database compared to the simpler version defined above in part a). (2 marks)

Question 3 [Total Marks: 20]

a) Explain the difference between a vertical slice of system functionality versus a horizontal slice. (2 marks)

b) Why is it preferable to use vertical slices over horizontal slices when developing software in an agile fashion? Give three reasons. (3 marks)

c) Imagine that you have been contracted to develop a *TV Guide* app for smart phones. The purpose of this app is to allow users to make informed choices about the movies that they watch on TV. Your clients, who have commissioned the application, have requested the following functionality:

- The app must run on both the Android and iOS platforms.
- The app must display all the film titles that will be shown on popular South African TV channels (SABC 1-3, eTV, M-Net and other DSTV channels) for the forthcoming week, along with:
  - the channel and time the film will be shown,
  - the film's age classification (eg. PG, 18VLN etc.),
  - the year the film was produced,
  - a synopsis of the plot,
  - the leading actors and actresses,
  - the critics rating from Rotten Tomatoes (a film reviews website), and
  - the user rating from Rotten Tomatoes.
- Films must be colour-coded according to their ratings to help users distinguish highly-rated films from poorly-rated films.
- Users must be able to select movies that they are interested in watching and the app must generate a reminder notification prior to the movie starting. The user should be able to select how far in advance the notification appears.

After conducting some investigation into the technical feasibility of constructing this application, you discover the following:

- The TV schedule for the forthcoming month for SABC 1-3, eTV and M-Net can be screen-scraped from `http://www.tvsite.co.za/`. Note, this site provides the entire schedule and not exclusively films but films are identified with a label. Age classifications are given for all shows on the schedule. There is a Python library called Beautiful Soup which facilitates the parsing of HTML pages.
- The TV schedule for DSTV channels can be accessed by parsing DSTV's JSON API available at `http://guide.dstv.com/`. Again, complete schedules are provided and not just films.
- Rotten Tomatoes has a public API which provides the year the film was produced, the film's synopsis, the cast, as well as the critics and user ratings.

i. Given the above functional requirements, identify and describe a subset of these requirements that would constitute a minimum viable product (MVP). Provide a justification that your MVP does provide some value to the user. (5 marks)

ii. Give four thin, vertical-slice user stories which are applicable to the MVP that you have identified above. Try to ensure that the slices are of a similar size and clearly state the acceptance criteria for each user story. (10 marks)

Question 4 [Total Marks: 12]

a) Explain what callbacks are *and* give a code example demonstrating the use of a callback. (3 marks)

```
function createfunc (i) {
  return function () {
    return `Item ${i}`
  }
}

function itemList () {
  const items = []
  let i = 0
  while (i < 3) {
    const item = createfunc(i)
    items.push(item)
    i++
  }

  return items
}

const myItems = itemList()
console.log(myItems.map(item => item()))
```

b) What will be the output of the above JavaScript code? (3 marks)

c) Give a detailed explanation of your answer. (6 marks)

[4 Questions — Available Marks: 77 — Full Marks: 70]

ELECTRICAL AND INFORMATION ENGINEERING
University of the Witwatersrand, Johannesburg
ELEN 4010: Software Development III

Reference Sheets

The following tables provide information on some of the methods which are available on JavaScript's `Array` and `String` objects. This information has been adapted from the MDN web docs.

1 `Array`

Mutator Methods	
`arr.pop()`	Removes the last element from an array and returns that element.
`arr.push(element1, ..., elementN)`	`element1` to `elementN` are added to the end of an array. Returns the new length of the array.
`arr.shift()`	Removes the first element from an array and returns that element.
`arr.unshift(element1, ..., elementN)`	`element1` to `elementN` are added to the front of an array. Returns the new length of the array.
`arr.reverse()`	Reverses the order of the elements of an array *in place*.
`arr.sort( compareFunction(firstEl, secondEl) )`	Sorts the elements of an array *in place* and returns the array. `compareFunction` defines the sort order. `firstEl` represents the first element for comparison. `secondEl` represents the second element for comparison. If `compareFunction` is omitted, the array is sorted according to each character's Unicode code point value, according to the string conversion of each element.
`arr.splice(start, deleteCount, item1, item2, ...)`	Changes the contents of an array by removing or replacing existing elements and/or adding new elements *in place*. `start` is the index at which to start changing the array (the index is 0-based). `deleteCount` is an integer indicating the number of old array elements to remove. If deleteCount is 0 or negative, no elements are removed. `item1`, `item2`, and so on are the elements to insert in the array, beginning at the `start` index. If you don't specify any elements, `splice` will only remove elements from the array. Returns an array containing the deleted elements. If no elements are removed, an empty array is returned.

Accessor Methods	
`arr.includes(searchElement, fromIndex)`	`searchElement` is the element to locate in the array. When comparing strings and characters, `includes` is *case sensitive*. `fromIndex` (optional) is the position in the array at which to begin searching for `searchElement`. Returns `true` if the element is found; `false` otherwise.
`arr.indexOf(searchElement, fromIndex)`	`searchElement` is the element to locate in the array. `fromIndex` (optional) is the index to start the search at. Returns the first index of the element in the array; `-1` if the element is not found.
`arr.lastIndexOf( searchElement, fromIndex)`	`searchElement` is the element to locate in the array. `fromIndex` (optional, defaults to `arr.length - 1`) is the index at which to start searching backwards. If `fromIndex` is omitted the whole array will be searched. Returns the last index of the element in the array; `-1` if not found.
`arr.join(separator)`	`separator` (optional) specifies a string to separate each pair of adjacent elements of the array. The separator is converted to a string if necessary. If omitted, the array elements are separated with a comma. If `separator` is an empty string, all elements are joined without any characters in between them. Returns a string with all array elements joined. If `arr.length` is 0, the empty string is returned.
`arr.slice(begin, end)`	The `slice` method returns a copy of a portion of an array as a new array object containing the elements selected from `begin` to `end` (the element at index `end` is not included). The original array will not be modified. If `end` is omitted, `slice` copies all elements to the end of the array.
`arr.toString()`	The `toString` method joins the array and returns a single string containing each array element separated by commas.

Iterative Methods	
These methods take as arguments a callback function which is called while processing the array. When these methods are called, the length of the array is noted, and any element added beyond this length from within the callback is not visited. The callback function has the following signature: `callback(element, index, array)`. `element` is the current element in the array being processed. `index` is the index of the current element being processed. `array` is the array which is being traversed.	
`arr.every(callback)`	The `every` method executes the provided callback function once for each element present in the array until it finds one where `callback` returns a false value. If such an element is found, the `every` method immediately returns `false`. Otherwise, `every` returns `true`. For an empty array, `every` always returns `true`.
`arr.some(callback)`	`some` executes the callback function once for each element present in the array until it finds one where `callback` returns a true value. If such an element is found, `some` immediately returns `true`. Otherwise, `some` returns `false`. For an empty array, `some` always returns `false`.

Iterative Methods	
`arr.filter(callback)`	The `filter` method creates a new array with all elements that pass the test implemented by the provided callback function. The callback function is a predicate, to test each element of the array. Return `true` to keep the element, `false` otherwise. If no elements pass the test, an empty array will be returned.
`arr.find(callback)`	The `find` method executes the callback function once for each element in the array until it finds one where `callback` returns a true value. If such an element is found, `find` immediately returns the element. In other words, the first element that satisfies the callback function is returned. If no elements are found, `find` returns `undefined`.
`arr.findIndex(callback)`	The `findIndex` method executes the callback function once for each element in the array until it finds one where `callback` returns a true value. If such an element is found, `findIndex` immediately returns the found element's index. If the callback never returns a true value or the array's length is 0, `findIndex` returns -1.
`arr.forEach(callback)`	`forEach` calls the provided callback function once for each element in the array. `forEach` does not mutate the array on which it is called (although `callback`, may do so). `forEach` always returns `undefined` and therefore is not chainable.
`arr.map(callback)`	`map` calls the provided callback function once for each element in the array and returns a new array containing the results. The `callback` is invoked only for array indexes which have assigned values. It is not invoked for empty slots in sparse arrays.

2 String

`str.charAt(index)`	Returns a new string consisting of the single character located at the specified index.
`str.includes(searchString, fromIndex)`	`searchString` is a string to be searched for within `str`. `fromIndex` (optional, defaults to 0) is the position within the string at which to begin searching for `searchString`. Returns `true` if the search string is found anywhere within the given string; otherwise, `false`. The `includes` method is *case sensitive*.
`str.indexOf(searchString, fromIndex)`	`searchString` is a string to be searched for within `str`. `fromIndex` (optional, defaults to 0) is the position within the string at which to begin searching for `searchString`. Returns the index of the first occurrence of `searchString`, or `-1` if not found. The search is *case sensitive*.
`str.substring(indexStart, indexEnd)`	`indexStart` is the index of the first character to include in the returned substring. `indexEnd` (optional, defaults to `str.length`) is the index of the first character to *exclude* from the returned substring. If `indexEnd` is omitted, the substring will extend to the end of `str`. Returns a new string containing the specified part of `str`.
`str.toLowerCase()`	Returns a new string representing `str` converted to lower case. `toLowerCase` does not modify `str` itself.
`str.toUpperCase()`	Returns a new string representing `str` converted to upper case. `toUpperCase` does not modify `str` itself.

3 Jest Test Framework

Test Structure

```
function sum (a, b) {
  return a + b
}

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3) // assertion
})
```

Basic Matchers

```
expect(42).toBe(42) // Strict equality (===)
expect(42).not.toBe(3) // Strict equality (!==)
expect([1, 2]).toEqual([1, 2]) // Deep equality
expect([1, 2]).not.toEqual([3, 4]) // Deep equality
```

Numbers

```
expect(2).toBeGreaterThan(1)
expect(1).toBeGreaterThanOrEqual(1)
expect(1).toBeLessThan(2)
```

```
expect(1).toBeLessThanOrEqual(1)

expect(4).toBe(4);  // toBe and toEqual are equivalent for numbers
expect(4).toEqual(4);

expect(0.1 + 0.2).toBeCloseTo(0.3, 5) // floating point numbers equal to 5 decimal places
```

Strings

```
expect('long string').toMatch('str') // string matches a substring
expect('pizza').not.toMatch('coffee')
```

Arrays

```
expect(['Alice', 'Bob', 'Eve']).toHaveLength(3)
expect(['Alice', 'Bob', 'Eve']).toContain('Alice')
```

Objects

```
expect({ a: 1 }).toHaveProperty('a') // object has the specified key
expect({ a: 1 }).toHaveProperty('a', 1) // object has the specified key-value pair
expect({ a: { b: 1 } }).toHaveProperty('a.b') // object has the nested key
expect({ a: 1, b: 2 }).toMatchObject({ a: 1 }) // object matches a subset of properties
```

SQL CHEAT SHEET

http://www.zentut.com/sql-tutorial/

SQL SELECT STATEMENTS

SELECT * FROM tbl
Select all rows and columns from table tbl

SELECT c1,c2 FROM tbl
Select column c1, c2 and all rows from table tbl

SELECT c1,c2 FROM tbl
WHERE conditions
ORDER BY c1 ASC, c2 DESC
Select columns c1, c2 with where conditions and from table tbl order result by column c1 in ascending order and c2 in descending order

SELECT DISTINCT c1, c2
FROM tbl
Select distinct rows by columns c1 and c2 from table tbl.

SELECT c1, aggregate(expr)
FROM tbl
GROUP BY c1
Select column c1 and use aggregate function on expression expr, group columns by column c1.

SELECT c1, aggregate(expr) AS c2
FROM tbl
GROUP BY c1
HAVING c2 > v
Select column c1 and c2 as column alias of the result of aggregate function on expr. Filter group of records with c2 greater than value v

SQL UPDATE TABLE

INSERT INTO tbl(c1,c2,...)
VALUES(v1,v2...)
Insert data into table tbl

INSERT INTO tbl(c1,c2,...)
SELECT c1,c2.. FROM tbl2
WHERE conditions
Insert data from tbl2 into tbl

UPDATE t
SET c1 = v1, c2 = v2...
WHERE conditions
Update data in table tbl

DELETE FROM tbl
WHERE conditions
Delete records from table tbl based on WHERE conditions.

TRUNCATE TABLE tbl
Drop table tbl and re-create it, all data is lost

SQL TABLE STATEMENTS

```
CREATE TABLE tbl(
        c1 datatype(length)
        c2 datatype(length)
        ...
        PRIMARY KEY(c1)
)
```
Create table tbl with primary key is c1

DROP TABLE tbl
Remove table tbl from database.

ALTER TABLE tbl
ADD COLUMN c1 datatype(length)
Add column c1 to table tbl

ALTER TABLE tbl
DROP COLUMN c1
Drop column c1 from table tbl

SQL JOIN STATEMENTS

SELECT * FROM tbl1
INNER JOIN tbl2 ON join-conditions
Inner join table tbl1 with tbl2 based on join-conditions.

SELECT * FROM tbl1
LEFT JOIN tbl2 ON join-conditions
Left join table tbl1 with tbl2 based on join-conditions.

SELECT * FROM tbl1
RIGHT JOIN tbl2 ON join-conditions
Right join table tbl1 with tbl2 based on join-conditions.

SELECT * FROM tbl1
RIGHT JOIN tbl2 ON join-conditions
Full outer join table tbl1 with tbl2 based on join-conditions.