EmSAT Achieve Computer Science- Python

Public Test Specification

Test Description: EmSAT Achieve Computer Science is a 120-minute computerized test that measures test takers' level of proficiency in Computer Science and determines their readiness for college. EmSAT Achieve Computer Science consists of two main Sections: Computer Science Theory and Problem Solving and Programming Practices. Test sections, questions, and options are randomized and timed by the test software. The computerized test is a timed test wherein the test clock is visible at all time to test takers.

Test Duration:	150 minutes	
Questions	100	
Content Areas:	**Section 1:** Computer Science Theory	1. Computing Systems and Networks 2. Data Analysis 3. Impacts of Computing
	Section 2: Problem Solving and Programming Practices	4. Algorithms and Programming - Python
Task Types:	Multiple Choice	
Test Language	English	
Calculators	Not Allowed	

EmSAT Achieve Computer Science	
Score	**Score Descriptors**
1500+	**High Proficiency:** students at this level are well-prepared for Physics courses at the university level.
1100-1475	**Proficient:** students at this level are at a satisfactory level of preparation to begin first-year Physics courses at the university level.
900-1075	**Borderline Proficient:** students at this level are minimally prepared for first-year Physics courses at the university level.
700-875	**Basic:** students at this level do not have sufficient mastery of prerequisite knowledge for first-year courses in Physics at the university level and may need some additional support.
500-675	**Needs Improvement:** students at this level need additional instructional support in basic Physics concepts and skills before beginning any first-year Physics courses.
< 500	**Little Knowledge of General Physics:** students at this level need intensive instructional support in basic Physics concepts and skills.

Publication Date: September 2021

EmSAT Achieve Computer Science- Python

Public Test Specification

Appendix 1: Content Areas

Below are the major sections and related content specifications that grade 12 students should be able to demonstrate mastery of in order to meet the expectations of this test.

Section 1: Computer Science Theory [40%]

This section tests the examinee knowledge in main computer science theory domains such as computer systems and network, data analysis, and impacts of computing.

1. **Computing Systems and Networks [25%]**
   Examinee should be able to:
   a. Identify the hardware components of a given computing system and describe the function of these components.
   b. Differentiate between different types of computing systems software and give examples on each software type (application software and system software).
   c. Demonstrate knowledge of how software control hardware and apply computing systems troubleshooting strategies on basic hardware and software problems.
   d. Design logic circuits and distinguish between the logic gates (AND, OR, NOT, XOR...etc.)
   e. Demonstrate knowledge of the computing trends (e.g., big data, machine learning, AI) and computing devices (e.g., microcontrollers, embedded systems ...etc.).
   f. Differentiate between different network types and recommend suitable network type for a given scenario.
   g. Differentiate between different types of network topologies and recommend suitable network topology for a given scenario.
   h. Identify different network hardware and software and demonstrate knowledge of their role in the network operation.
   i. Demonstrate knowledge of network architecture and task allocation between network hosts (Client-Server Model and Peer-to-Peer Model).
   j. Identify the network security issues and threats and apply the network security principles in network design.
   k. Demonstrate knowledge of network communication layers models and identify each layer functions and the protocols serving each layer.
   l. Identify different types of addresses and explain their role within one network communication or between different networks communication.
   m. Compare guided (wired) and unguided (wireless) transmission media in term of cost, reliability, and security.
   n. Identify the factors that affect the network performance and distinguish between the different components of nodal delay.
   o. Identify security measures designed to protect computer networks and describe vulnerabilities that the various types of cyber threats can exploit.

2. **Data Analysis [10%]**

Examinee should be able to:

a. Identify different data collection methods and apply these methods for locating and collecting a variety of data sets.
b. Analyze and identify patterns in a variety of data sets.
c. Identify different methods to store data and manipulate them and demonstrate knowledge of issues related to data security.
d. Identify different numbering systems and convert between numbering system to another.
e. Use the binary numbering system to represent different types of data in computers such as sound, image and text.
f. Select appropriate representations of data (e.g., charts, graphs, network diagrams, flowcharts) and use computers to model and simulate different real-life processes and phenomena.

3. **Impacts of Computing [5%]**

Examinee should be able to:

a. Distinguish between responsible and irresponsible use of technology and give example for each use.
b. Identify the UAE cyber law and apply the UAE cyber law on real life scenario.
c. Identify the positive and the negative effect of technology on different society aspects such as education, economy, innovation, collaboration, workplaces, and health.
d. Explain the concept of digital divide and demonstrate knowledge of issues related to the equitable use of technology.

Section 2: Problem Solving and Programming Practices [60%]

This section tests the examinee knowledge and skills in problem solving techniques and tests whether the examinee is able to use programming skills as a tool to solve computational problems.

4. **Algorithms and Programming [60%]**

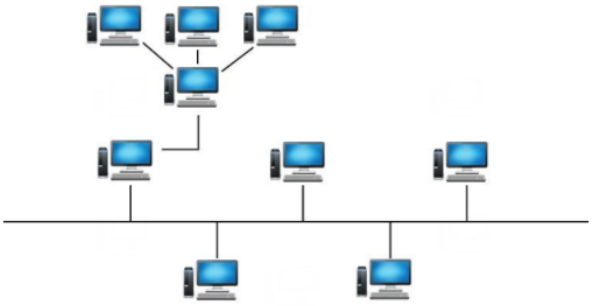
   Examinee should be able to:

   a. Break programming specifications into steps and use different algorithm representations such as pseudocodes and flowcharts to represent algorithms as first stage before coding.
   b. Evaluate and compare algorithms in term of their efficiency, simplicity, complexity, and clarity and suggest modifications to improve algorithms functionality.
   c. Apply the pillars of computational thinking as a process to solve a computational problem and select appropriate method to a given context.
   d. Create different types of variables (data types: integer, double, string...etc.) and differentiate between variables and constants in term of their roles and manipulation.
   e. Distinguish between different operators (arithmetic, logical and relational) and evaluate simple and compound expressions.
   f. Create different static data structures and perform different operations (update, swap, research...etc.) on them in order to manipulate their elements or extract information.
   g. Read and write data from external data structures such as files and decide when it is appropriate to use external data structure.
   h. Create different dynamic data structures and perform different operations (update, swap, research...etc.) on them in order to manipulate their elements or extract information.
   i. Program using Procedure-Oriented Programming (POP) and create different types of functions based on whether they accept arguments and/or return values.
   j. Program using Object-Oriented Programming (OOP) and be able to apply the features of the OOP such as inheritance, encapsulation, abstraction, and polymorphism.
   k. Combine sequence steps of instructions in order to achieve a specific task.
   l. Distinguish between different selection statements (If Statement, If-Else Statement, Nested If-Statement, Switch/Case) and select the appropriate selection statement based on the problem given.
   m. Distinguish between different iteration statements (For Loop, While Loop, Do-While Loop) and select the appropriate iteration statement based on the problem given.
   n. Compare and contrast different high-level programming languages and identify the main components of the programming environment.
   o. Combine all programming constructs (sequence, selection, and iteration) and components (variables, control structures, operators, functions...etc.) together in order to build a program that meets certain design specifications.
   p. Identify different types of programming errors (runtime, syntax and logical) and apply different testing techniques to ensure program correctness.
   q. Apply programming best practices when coding and produce well documented program that is easy to read, reuse and maintain.

EmSAT Achieve Computer Science- Python
Public Test Specification

Appendix 2: Sample Items

1. What is this network topology of the following computer network?



A. a hybrid network topology

B. a linear network topology

C. an interconnected network topology

D. a branched network topology

2. Which of the following is **not** an IP address?

A. 260.1.0.5

B. 244.193.55.13

C. 8.8.8.8

D. 1.0.0.1

3. Which of the following is **not** an operating system?

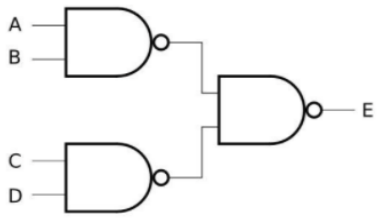
A. MS Office 365

B. Linux

C. Android

D. Macintosh

4. Which of the following Boolean expressions is equivalent to the following digital logic circuit?



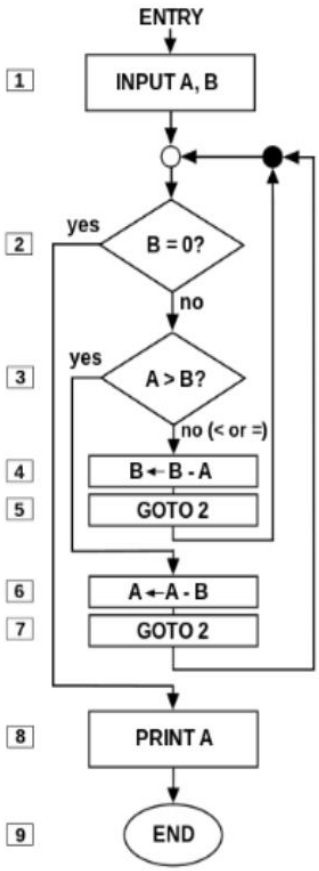
A. $E = A.B + C.D$

B. $E = (\overline{A.B} + \overline{C.D})$

C. $E = (\overline{A.B + CD})$

D. $E = \overline{((A + B).(C + D))}$

5. Given the following algorithm (flowchart), what is the output of the last statement, **PRINT A**, if the inputs are A = 78, and B = 12?



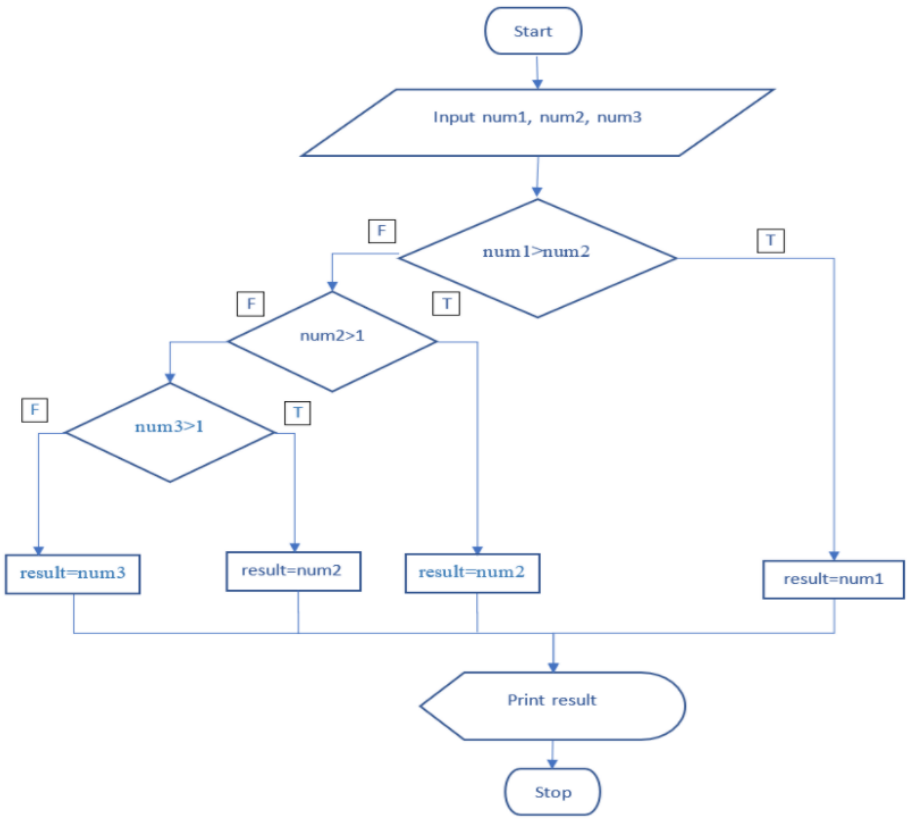
A. 6

B. 7

C. 66

D. 78

6. What is the output of the following algorithm (flowchart) if the inputs are num1 = -1, num2 = 0, and num3 = 6?



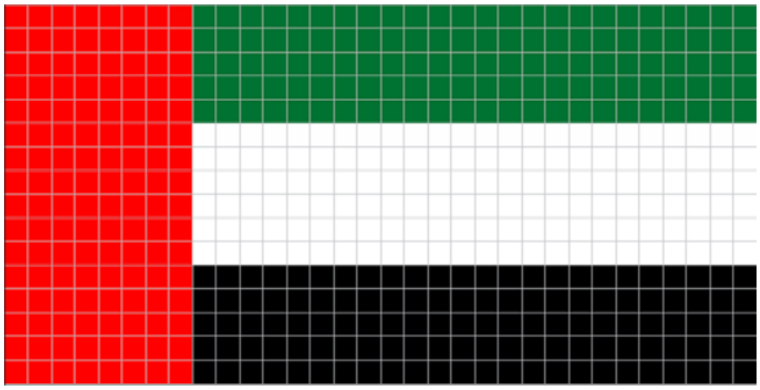
A. 0

B. -1

C. 6

D. 1

7. What is the size (in bytes) of the following 32×16 image with 16-bit color code?



A. 1024 bytes

B. 512 bytes

C. 96 bytes

D. 128 bytes

8. What is the output of the following code?

```
A=[1,4,2,0,3]
temp = A[0]

for i in range(len(A)-1):
    A[i]=A[i+1]

A[len(A)-1]=temp

for i in range(len(A)):
   print (A[A[i]],end = '')
```

- 10432
- 42031
- 30241
- 31420

9. What is the output of the following code?

```
s="UAE2019"
t=""
for i in range(len(s)):
    t = s[i] + t
print(t)
```

- 9102EAU
- 2019UAE
- UAE2019
- UAE20199102EAU

10. Given the following recursive function:

```
def recursive(n):
    if (n <= 2):
        return n
    else:
        return (n + recursive(n-1) + recursive(n-2))
```

What is the output of the following function call?

```
print( recursive(5))
```

- 23
- 19
- 16
- 0

11. What is the output of the following statement?

```
print((1.0 - 6 * 4 / 5 ) /  (17 % 5))
```

- −1.9
- −1
- −2
- −1.5

12. What is the output of the following statement?

```
print(2**3+28%9*5)
```

- ○ 13
- ○ 46
- ○ 54
- ○ 24

Answer Key:

1. A
2. A
3. A
4. A
5. A
6. A
7. A
8. A
9. A
10. A
11. A
12. A

Appendix 2: EmSAT Pseudocode Guide

Section 1: Variables and Data Types

Action	Rule	Example
Variable declaration	**int** variable_name **double** variable_name **char** variable_name **string** variable_name	**int** x **double** weight **char** vitamin **string** name
Variable declaration and initialization	**data_type** variable name ← value	**int** x ← 3 **char** vitamin ← 'A' **string** name ← "Wafa"
Passing value to a variable	variable ← value	x ← 6 name ← "Wafa" vitamin ← 'C'
Incrementing the value of a variable	variable ← variable +1	x ← x+1
Decrementing the value of a variable	variable ← variable -1	x ← x-1
Moving the value of a variable to another variable	Variable_2 ← variable_1	y ← x

Section 2: Static and Dynamic Data Strucutres

Data Structure		Rule	Example
Static 1D Array	Declaration and Initialization	**data_type** array_name [] ← {element 1, element 2, element N}	**int** grade [] ← {88, 83, 99} **double** temp [] ← {33.2, 37.1, 39.2} **string** name [] ← { "Wafa", "Nafla", "Rola" }
	Update	array_name [index]← value	**int** grade [] ← {88, 83, 99} grade [1]← 84 // replace 83 with 84

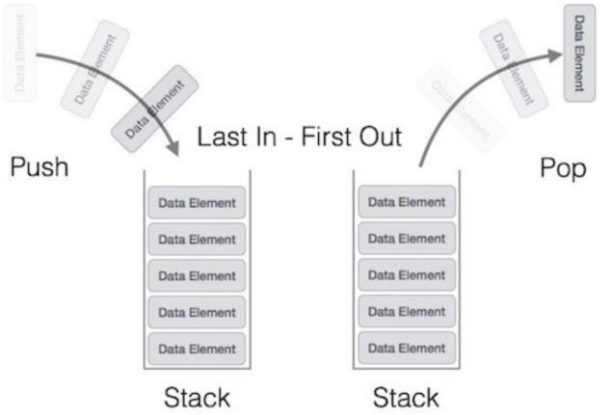
	Search	**data_type** array_name [] ← {element 1, element 2, element N}
		FOR (**int** i ← 0, i<N, i ← i+1)
		IF (array_name [i] == value)
		PRINT "found"
		ELSE
		PRINT "not found"
		END IF
		END FOR
			int grade [] ← {88, 83, 99}
			FOR (**int** i ← 0, i<3, i ← i+1)
			IF (grade [i] == 83)
			PRINT "found"
			ELSE
			PRINT "not found"
			END IF
			END FOR

	Swap		array_name [index_target]← array_name [index_source]	**int** grade [] ← {88, 83, 99} grade [1]← grade [2] // swap 83 with 99

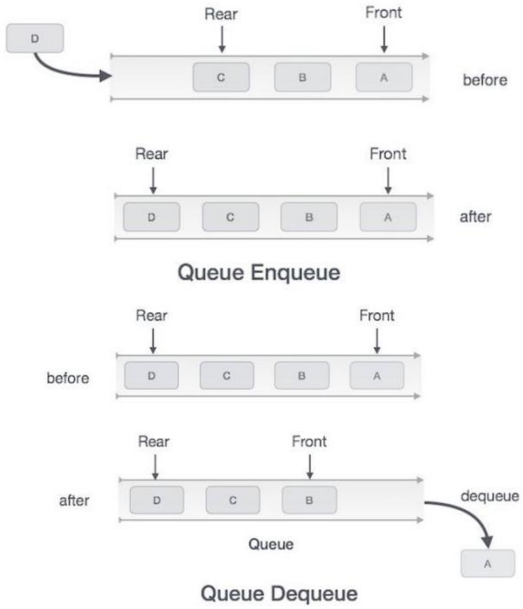
Static 2D Array	**Declaration and Initialization**		**data_type** array_name [][]
			FOR (**int** i ← 1, i<N, i ← i+1)
			  FOR (**int** j ← 1, (j<N), j ← j+1)
			  array_name [i][j]← value
			  END FOR
			END FOR
				int 2D_multiplication [][]
				FOR (**int** i ← 1, i<10, i ← i+1)
				  FOR (**int** j ← 1, j<10, j ← j+1)
				  2D_multiplication [i][j]← i*j
				  END FOR
				END FOR

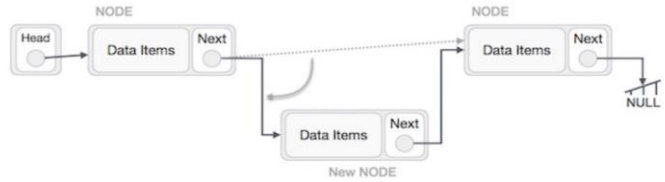
	Update		**data_type** array_name [][]
			FOR (**int** i ← 1, i<N, i ← i+1)
			  FOR (**int** j ← 1, j<N, j ← j+1)
			  array_name [i][j]← value
			  END FOR
			END FOR
				int 2D_multiplication [][]
				FOR (**int** i ← 1, i<10, i ← i+1)
				  FOR (**int** j ← 1, j<10, j ← j+1)
				  2D_multiplication [i][j]← i*j
				  END FOR
				END FOR

	Search		**data_type** array_name [][]
			FOR (**int** i ← 1, i<N, i ← i+1)
			  FOR (**int** j ← 1, j<N, j ← j+1)
			  IF (array_name [i][j]← value)
			  PRINT “found”
			  ELSE
			  PRINT “not found”
			  END IF
			  END FOR
			END FOR
				int 2D_multiplication [][]
				FOR (**int** i ← 1, i<N, i ← i+1)
				  FOR (**int** j ← 1, j<N, j ← j+1)
				  IF (2D_multiplication [i][j]← 30)
				  PRINT “found”
				  ELSE
				  PRINT “not found”
				  END IF
				  END FOR
				END FOR

Dynamic Data Structure	**Stack**	**Basic Operations**	**PUSH** () – Pushing (storing) an element on the stack.
			IF stack **isFULL**
			  RETURN NULL
			END IF
			Top ← top + 1
			stack[top] ← data
			
			POP () – Removing (accessing) an element from the stack.
			IF stack **isEMPTY**
			  RETURN NULL
			END IF
			data ← stack[top]
				

			top ← top - 1	

	Queue	**Basic Operations**	**ENQUEUE ()** – add (store) an item to the queue.
			IF queue **isFULL**
			  RETURN OVERFLOW
			END IF
			Rear ← rear + 1
			queue[rear] ← data
			DEQUEUE () – remove (access) an item from the queue.
			IF queue **isEMPTY**
			  RETURN UNDERFLOW
			END IF
			data ← queue[front]
			front ← front + 1
				

	Linked List	**Insertion Operation**	NewNode.Next → RightNode LeftNode.Next → NewNode	
		Deletion Operation	LeftNode.Next → TargetNode.Next TargetNode.Next → NULL	

Section 3: Operators and Expressions

Operator	Rule	Example
Arithmetic	+, −, *, %, /, ^ **Note:** / indicates floating point division unless stated otherwise	**int** r formula ← 2*PI*r^2

Relational	>, <, =, ≠, ≤, ≥,	**int** value_1
		int value_2
		READ value_1, value_2
		IF (value_1 > value_2)
		  PRINT "value_1 is bigger than value_2"
		ELSE
		  PRINT "value_1 is smaller than value_2"
		END IF

Logical	AND, OR, NOT	**int** x
		READ x
		IF (x ≠ 0 and x>0) // print the value if its zero or positive
		  PRINT x
		ELSE
		  PRINT "entry is negative"

		END IF

Section 4: Iteration

Loop	Rule	Example
While Loop	counter initialization **WHILE** (condition) statement/s increment counter **END WHILE**	**int** value ←1 **WHILE** (value ≠ 6) **PRINT** value value ← value+1 **END WHILE**
Do while	counter initialization **DO** statement/s increment counter **WHILE** (condition)	**int** i ← 1 **DO** **PRINT** "Hello World!" i ← i+1 **WHILE** (i<10)
For Loop	**FOR** (initialization, (condition), increment) statement/s **END FOR**	**FOR** (**int** i ← 0; (i <10); i ← i+1) **PRINT** i **END FOR**
Nested For Loop	**FOR** (initialization, (condition), increment) **FOR** (initialization, (condition), increment) statement/s **END FOR** **END FOR**	**FOR** (**int** i ← 1, (i<10), i ← i+1) **FOR** (**int** j ← 1, (j<10), j ← j+1) **PRINT** i+j **END FOR** **END FOR**

Section 5: Selection

Selection	Rule	Example
If Statement	**IF** (condition) statement/s **END IF**	**int** value **READ** value **IF** (value ≠ 0) **PRINT** value **END IF**
If Else Statement	**IF** (condition) statement/s **ELSE** statement/s	**int** value_1 **int** value_2 **READ** value_1 **READ** value_2

	END IF	**IF** (value_1 > value_2)
		  PRINT "value_1 is bigger than value_2"
		ELSE
		  PRINT "value_1 is smaller than value_2"
		END IF

Nested If Statement	**IF** (condition)
	  statement/s
	ELSE
	  IF (condition)
	  statement/s
	  ELSE
	  IF (condition)
	  statement/s
	  ELSE
	  statement/s
	  END IF
	  END IF
	END IF
		int grade
		READ grade
		IF (grade ≥ 90)
		  PRINT "grade is A"
		ELSE
		  IF (grade ≥ 80)
		  PRINT "grade is B"
		  ELSE
		  IF (grade ≥ 70)
		  PRINT "grade is C
		  ELSE
		  PRINT "grade is F"
		  END IF
		  END IF
		
		END IF

Switch	**data_type** value
	READ value
	  CASE 1: (condition 1)
	  statement/s
	  CASE 2: (condition 2)
	  statement/s
	  CASE 3: (condition 3)
	  statement/s
	  CASE N: (condition N)
	  statement/s
	  DEFAULT
	  statement/s
	END CASE
		int grade
		READ grade
		  CASE 1: (grade ≥ 100)
		  PRINT "perfect score"
		  CASE 2: (grade > 89)
		  PRINT "grade is A"
		  CASE 3: (grade > 79)
		  PRINT "grade is B"
		  CASE 4: (grade > 69)
		  PRINT "grade is C"
		  CASE 5: (grade > 59)
		  PRINT "grade is D"
		  DEFAULT
		  PRINT "grade is F"
		END CASE

Section 5: Procedure-Oriented Programming

Function Body		Rule	Example
returns	**arguments**		
x	x	**void FUNCTION** function_name () statement/s **END FUNCTION** function_name	**void FUNCTION** greetings () **PRINT** "Hello" **END FUNCTION** greetings
x	√	**void FUNCTION** function_name (arg1, arg2….) statement/s **END FUNCTION** function_name	**void FUNCTION** greetings (customer_name) **PRINT** "Hello, customer_name" **END FUNCTION** greetings
√	x	**data_type FUNCTION** function_name () statement/s **RETURN** value **END FUNCTION** function_name	**string FUNCTION** myname () name ← "Wafa" **RETURN** name **END FUNCTION** myname
√	√	**data_type FUNCTION** function_name (arg1, arg2…) statement/s **RETURN** value **END FUNCTION** function_name	**int FUNCTION** multiplication (value_1, value_2) result ← value_1 * value_2 **RETURN** result **END FUNCTION** multiplication
Function Call		**Rule**	**Example**
returns	**arguments**		
x	x	function_name ()	greetings ()
x	√	function_name (arg1, arg2, argN)	greetings (Wafa)
√	x	function_name ()	myname ()
√	√	function_name (arg1, arg2, argN)	multiplication (10, 3)

Section 6: Object-Oriented Programming

Actions	Rule	Example
Class declaration	**CLASS** class_name **variable declarations**	**CLASS** student **string** name

	functions **END CLASS** class_name	**double** GPA **int** Grade **void** register () **void** drop () **END CLASS** student
Object Creation	**Object_name** class_name	**std1** student

Section 7: Others

Boolean	TRUE, FALSE
Null	NULL
Comments	// type the comments here
Placeholder for missing code	/* missing code */ /* condition */
Keywords	READ RETURN PRINT DEFAULT SIZE LENGTH CASE PI Void BREAK TRUE FALSE WRITE SQUARE
Data Types	int double char string float