**AP Computer Science Principles Exam**

**Reference Sheet**

As AP Computer Science Principles does not designate any particular programming language, this reference sheet provides instructions and explanations to help students understand the format and meaning of the questions they will see on the exam. The reference sheet includes two programming formats: text based and block based.

Programming instructions use four data types: numbers, Booleans, strings, and lists.

Instructions from any of the following categories may appear on the exam:

* Assignment, Display, and Input
* Arithmetic Operators and Numeric Procedures
* Relational and Boolean Operators
* Selection
* Iteration
* List Operations
* Procedures
* Robot

**Assignment, Display, and Input**

Code 1:

Text:

a ← expression

Block:

“a ← expression” with a block around it

Explanation 1:

Evaluates expression and assigns the result to the variable a.

Code 2:

Text:

DISPLAY (expression)

Block:

“DISPLAY” in a large box. “expression” in a smaller box next to the word “DISPLAY” in the larger box.

Explanation 2:

Displays the value of expression, followed by a space.

Code 3:

Text:

INPUT ()

Block:

“INPUT”

Exlanpation3:

Accepts a value from the user and returns it.

**Arithmetic Operators and Numeric Procedures**

Code 4:

Text and Block:

a + b

a - b

a \* b

a / b

Explanation 4:

The arithmetic operators +, -, \*, and / are used to perform arithmetic on a and b.

For example, 3 / 2 evaluates to 1.5.

Code 5:

Text and Block:

a MOD b

Explanation 5:

Evaluates to the remainder when a is divided by b. assume that a and b are positive integers.

For example, 17 MOD 5 evaluates to 2.

Code 6:

Text:

RANDOM (a, b)

Block:

“RANDOM” next to “a, b” in a small box

Explanation 6:

Evaluates to a random integer from a to b, including a and b.

For example, RANDOM (1, 3) could evaluate to 1, 2, or 3.

**Relational and Boolean Operators**

Code 7:

Text and Block:

a = b

a ≠ b

a > b

a < b

a ≥ b

a ≤ b

Explanation 7:

The relational operators =, ≠, >, <, ≥, and ≤ are used to test the relationship between two variables, expressions, or values.

For example, a = b evaluates to true if a and b are equal; otherwise, it evaluates to false.

Code 8:

Text:

NOT condition

Block:

“NOT” next to “condition” in a small oval

Explanation 8:

Evaluates to true if condition is false; otherwise evaluates to false.

Code 9:

Text:

condition1 AND condition2

Block:

“condition1” in a small oval next to “AND” next to “condition2” in a small oval

Explanation 9:

Evaluates to true if both condition1 and condition2 are true; otherwise, evaluates to false.

Code 10:

Text:

condition1 OR condition2

Block:

“condition1” in a small oval next to “OR” next to “condition2” in a small oval

Explanation 10:

Evaluates to true if condition1 is true or if condition2 is true or if both condition1 and condition2 are true; otherwise, evaluates to false.

**Selection**

Code 11:

Text:

IF (condition)

{

 <block of statements>

}

Block:

Large box around all following text: “IF” next to “condition” in a small oval. “block of statements” in a small double oval on a new line under the “condition” oval.

Explanation 11:

The code in block of statements is executed if the Boolean expression condition evaluates to true; no action is taken if condition evaluates to false.

Code 12:

Text:

IF (condition)

{

 <first block of statements>

}

 ELSE

{

 <second block of statements>

}

Block:

Large box around all following text: “IF” next to “condition” in a small oval. “first block of statements” in a small double oval on a new line under the “condition” oval. “ELSE” on a new line, lined up to the left margin. “second block of statements” in a small double oval on a new line, in line with the other double oval.

Explanation 12:

The code in first block of statements is executed if the Boolean expression condition evaluates to true; otherwise, the code in second block of statements is executed.

**Iteration**

Code 13:

Text:

REPEAT n TIMES

{

 <block of statements>

}

Block:

Large box around the following text: “REPEAT n TIMES.” On a new line and indented, “block of statements” in a double oval.

Explanation 13:

The code in block of statements is executed n times.

Code 14:

Text:

REPEAT UNTIL (condition)

{

<block of statements>

}

Block:

Large box around the following text: “REPEAT UNTIL” next to “condition” in a small oval. On a new line and indented, “block of statements” in a double oval.

Explanation 14:

The code in block of statements is repeated until the Boolean expression condition evaluates to true.

**List Operations**

For all list operations, if a list index is less than 1 or greater than the length of the list, an error message is produced and the program terminates.

Code 15:

Text:

list[i]

Block:

“list” with “i” in a small box connected to it.

Explanation 15:

Refers to the element of list at index i. The first element of list is at index 1.

Code 16:

Text:

list[i] ← list[j]

Block:

A larger oval around the following text: “list” next to “i” in a box. an arrow pointing at “listi” from “list” with “j” in a small box.

Explanation 16:

Assigns the value of list[j] to list[i].

Code 17:

Text:

list ← [value1, value2, value3]

Block:

A larger oval around the following text: “list” an arrow pointing at “list” from “value1, value2, value3” in a small box.

Explanation 17:

Assigns value1, value2, and value3 to list[1], list[2], and list[3], respectively.

Code 18:

Text:

FOR EACH item IN list

{

 <block of statements>

}

Block:

A large box around the following text: “FOR EACH item IN list.” “block of statements” indented in a double oval on a new line.

Explanation 18:

The variable item is assigned the value of each element of list sequentially, in order from the first element to the last element.

The code in block of statements is executed once for each assignment of item.

Code 19:

Text:

INSERT (list, i, value)

Block:

A box around the following test: “INSERT” next to “list, i, value” in a small box.

Explanation 19:

Any values in list at indices greater than or equal to i are shifted to the right. The length of list is increased by 1, and value is placed at index i in list.

Code 20:

Text:

APPEND (list, value)

Block:

A box around the following text: “APEND” next to “list, value” in a small box.

Explanation 20:

The length of list is increased by 1, and value is placed at the end of list.

Code 21:

Text:

REMOVE (list, i)

Block:

A box around the following text: “REMOVE” next to “list, i” in a small box.

Explanation 21:

Removes the item at index i in list and shifts to the left any values at indices greater than i. The length of list is decreased by 1.

Code 22:

Text:

LENGTH (list)

Block:

 “LENGTH” next to “list” in a small box.

Explanation 22:

Evaluates to the number of elements in list.

**Procedures/Functions/Actions**

Code 23:

Text:

PROCEDURE name ( parameter1,

 parameter2, ...)

{

 <instructions>

}

Block:

Large box around the following text: “PROCEDURE name” next to “parameter1” and on a new line “perameter2,…” in a box. On a new line, “instructions” indented and in a double oval.

Explanation 23:

A procedure, name, takes zero or more parameters. The procedure contains programming instructions.

Code 24:

Text:

PROCEDURE name ( parameter1,

 parameter2, ...)

{

 <instructions>

 RETURN (expression)

}

Block:

Large box around the following text: “PROCEDURE name” next to “parameter1” and on a new line “perameter2,…” in a box. Large oval around the following text: on a new line, “instructions” indented and in oval. On a new line, in an oval “RETURN” next to “expression” in a box.

Explanation 24:

A procedure, name, takes zero or more parameters. The procedure contains programming instructions and returns the value of expression. The RETURN statement may appear at any point inside the procedure and causes an immediate return from the procedure back to the calling program.

**Robot**

If the robot attempts to move to a square that is not open or is beyond the edge of the grid, the robot will stay in its current location and the program will terminate.

Code 25:

Text:

MOVE\_FORWARD ()

Block:

“MOVE\_FORWARD” in a box

Explanation 25:

The robot moves one square forward in the direction it is facing.

Code 26:

Text:

ROTATE\_LEFT ()

Block:

“ROTATE\_LEFT” in a box

Explanation 26:

The robot rotates in place 90 degrees counterclockwise (i.e., makes an in-place left turn).

Code 27:

Text:

ROTATE\_RIGHT ()

Block:

“ROTATE\_RIGHT” in a box

Explanation 27:

The robot rotates in place 90 degrees clockwise (i.e., makes an in-place right turn).

Code 28:

Text:

CAN\_MOVE (direction)

Block:

“CAN\_MOVE” next to “direction” in a box

Explanation 28:

Evaluates to true if there is an open square

one square in the direction relative to where

the robot is facing; otherwise evaluates to

false. The value of direction can be left,

right, forward, or backward.

**Important Differences Between Quorum and the Reference Language**

There are some differences between Quorum and the Reference Language. Some of the key differences are:

**Assignment Operator**

Quorum:

text frog = “bullfrog”

number siblings = 5

Reference Language

a ← expression

**Input**

Quorum:

text question = input(“ “)

Reference Language:

INPUT ()

**Output**

Quorum:

output “expression”

say “expression”

Reference Language:

DISPLAY (expression)

**Conditional Operators**

Quorum:

a = b

a =/ b

a > b

a < b

a >= b

a <= b

Reference Language:

a = b

a ≠ b

a > b

a < b

a ≥ b

a ≤ b

**Boolean Expressions**

Quorum:

if condition = false

Reference Language:

NOT condition

Quorum:

if condition1 = 2 and condition2 = 4

Reference Language:

condition1 AND condition2

Quorum:

if condition2 = 2 or condition2 = 4

Reference Language:

condition1 OR condition2

**Repeat Loops**

Quorum:

repeat n times

 block of statements

end

Reference Language:

REPEAT n TIMES

{

 <block of statements>

}

Quorum:

repeat until condition

 block of statements

end

Reference Language:

REPEAT UNTIL (condition)

{

 <block of statements>

}

**Procedure/Function/Action**

Quorum:

action Name (parameter1, parameter2, …)

 instructions

end

Reference Language:

PROCEDURE name (parameter1,

 Parameter2,…)

{

 <instructions>

}

**Turtle Moves**

Quorum:

moveForward()

Reference Language:

MOVE\_FORWARD ()

Quorum:

turnLeft()

Reference Language

ROTATE\_LEFT ()

Quorum:

No pre-made action

Reference Language:

ROTATE\_RIGHT ()

Quorum:

There is no such function in Quorum. CAN\_MOVE returns true if the turtle can move in the direction requested (i.e. the square in the direction given is white and not off the grid)

Reference Language:

CAN\_MOVE (direction)