**IB Pseudocode Syntax**

Basic instructions

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Syntax** | **Description** | **Examples** |
| Assign | *variable* = *value* | Assigns a value to the variable | a = 1 |
| Input | input *variable* | Inputs the variable | input a |
| Output | output *variable/expression* | Outputs a value of the variable or expression | output a  output “hello”  output 2 + 2 |
| Create | create *type variable* | Creates the variable with standard value of given type | create Boolean a  create Number b  create String c |
| Delete | delete *variable* | Deletes the variable | delete a |

Conditions

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Syntax** | **Description** | **Examples** |
| If | if *condition* then | Indicates the start of a condition block and states the first condition | if a = 1 then |
| Else if | else if *condition* then | States an additional condition | else if a = 2 then |
| Else | else | Indicates the start of the part of a condition block which will be executed if all conditions above are false | else |
| End if | end if | Indicates the end of a condition block | end if |

Loops

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Syntax** | **Description** | **Examples** |
| While loop | loop while *condition* | Executes a loop block while the condition is true | loop while a < 5 |
| Until loop | loop until *condition* | Executes a loop block until the condition is true | loop until a = 5 |
| For loop | loop *variable* from *start value* to *end value* | Executes a loop block for every value of the variable between start value and end value | loop a from 1 to 5 |
| loop for *variable* from *start value* to *end value* | loop for a from 1 to 5 |
| End loop | end loop | Indicates the end of a loop block | end loop |

Operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Syntax** | **Description** | **Examples** |
| Equal | *value1* = *value2* | Checks is the first value equal to the second | a = 1 |
| Not equal | *value1* != *value2*  *value1* <> *value2* | Checks is the first value not equal to the second | a != 1  a <> 1 |
| Greater | *value1* > *value2* | Checks is the first value greater than the second | a > 1 |
| Greater or equal | *value1* >= *value2* | Checks is the first value greater or equal to the second | a >= 1 |
| Less | *value1* < *value2* | Checks is the first value less than the second | a > 1 |
| Less or equal | *value1* <= *value2* | Checks is the first value less or equal to the second | a >= 1 |
| Not | NOT *value1* | Executes logical or bitwise NOT for the value | NOT a |
| And | *value1* AND *value2* | Executes logical or bitwise AND for the first and the second values | a AND 1 |
| Or | *value1* OR *value2* | Executes logical or bitwise OR for the first and the second values | a OR 1 |
| Xor | *value1 X*OR *value2* | Executes bitwise XOR for the first and the second values | a XOR 1 |
| Addition | *value1* + *value2* | Adds the first and the second values | a + 1 |
| Subtraction | *value1* - *value2* | Subtracts the first and the second values | a - 1 |
| Multiplication | *value1* \* *value2* | Multiplies the first and the second values | a \* 1 |
| Division | *value1* / *value2* | Divides the first and the second values | a / 1 |
| Modulo | *value1* mod *value2* | Gets modulo of the first and the second values | a mod 1 |
| Integer division | *value1* div *value* | Gets integer part of the division of the first and the second values | a div 1 |

Functions

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Syntax** | **Description** | **Examples** |
| Function | function *name*(*arg1, …)* | Indicates the start of a function block with name and arguments | function f(a, b) |
| Return | return *variable/expression* | Returns value or expression from function | return a  return “hello”  return 2 + 2 |
| End function | end function | Indicates the end of a function block | end function |
| Run function | *name*(*arg1, …)* | Runs a function block with given name and arguments | f(1, 2) |

Procedures

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Syntax** | **Description** | **Examples** |
| Procedure | procedure *name*(*arg1, …)* | Indicates the start of a procedure block with name and arguments | procedure p(a, b) |
| End procedure | end procedure | Indicates the end of a procedure block | end procedure |
| Run procedure | *name*(*arg1, …)* | Runs a procedure block with given name and arguments | p(1, 2) |

Basic data types

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Syntax** | **Description** | **Examples** |
| Boolean | *variable* = true  *variable* = false | Boolean type that can contain only true or false values | a = true  b = false |
| create Boolean *variable* | create Boolean a |
| Boolean *variable* | Boolean a |
| Number | *variable* = *0* | Number type that can contain any number value | a = 0 |
| create Number *variable* | create Number a |
| Number *variable* | Number a |
| String | *variable* = “text” | String type that can contain any text | a = “hello” |
| create String *variable* | create String a |
| String *variable* | String a |

Arrays

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Syntax** | **Description** | **Examples** |
| Create array | create Array *name* | Creates an empty array with given name | create Array a |
| Array *name* | Array a |
| Get item | *name*[*index*] | Returns an item with given index from the array | a[0] |
| Set item | *name*[*index*] = *value* | Assigns a value to given index from the array | a[0] = 1 |
| Array size/length | *name*.size() | Returns a size/length of the array | a.size() |
| *name*.length() | a.length() |
| Assign array | *name* =[*val1, val2, …*] | Assigns an array with given values to a variable | a = [1, 2, 3] |

Dictionaries

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Syntax** | **Description** | **Examples** |
| Create dictionary | create Dictionary *name* | Creates an empty dictionary with given name | create Dictionary a |
| Dictionary *name* | Dictionary a |
| Get item | *name*[“*key”*] | Returns an item with given key from the dictionary | a[“a”] |
| Set item | *name*[“*key”*] = *value* | Assigns a value to given key from a dictionary with given name | a[“a”] = 1 |

Collections

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Syntax** | **Description** | **Examples** |
| Create collection | create Collection *name* | Creates an empty collection with given name | create Collection a |
| Collection *name* | Collection a |
| Add item | *name*.addItem(*value*) | Adds a value to the end of the collection | a.addItem(1) |
| Get next | *name*.getNext() | Returns next value from the collection | a.getNext() |
| Reset next | *name*.resetNext() | Resets next element of the collection | a.resetNext() |
| Has next | *name*.hasNext() | Checks does the collection have next element | a.hasNext() |
| Is empty | *name*.isEmpty() | Check does the collection contains elements | a.isEmpty() |
| Collection size/length | *name*.size() | Returns a size/length of the collection | a.size() |
| *name*.length() | a.length() |
| Assign collection | *name* ={*val1, val2, …*} | Assigns a collection with given values to a variable | a = {1, 2, 3} |

Stacks

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Syntax** | **Description** | **Examples** |
| Create stack | create Stack *name* | Creates an empty stack with given name | create Stack a |
| Stack *name* | Stack a |
| Push | *name*.push(*value*) | Adds a value to the stack | a.push(1) |
| Pop | *name*.pop() | Gets a value from the stack | a.pop() |
| Is empty | *name*.isEmpty() | Check does the stack contains elements | a.isEmpty() |
| Stack size/length | *name*.size() | Returns a size/length of the stack | a.size() |
| *name*.length() | a.length() |

Queues

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Syntax** | **Description** | **Examples** |
| Create queue | create Queue *name* | Creates an empty queue with given name | create Queue a |
| Queue *name* | Queue a |
| Enqueue | *name*.enqueue(*value*) | Adds a value to the queue | a.enqueue(1) |
| Dequeue | *name*.dequeue() | Gets a value from the queue | a.dequeue() |
| Is empty | *name*.isEmpty() | Check does the queue contains elements | a.isEmpty() |
| Queue size/length | *name*.size() | Returns a size/length of the queue | a.size() |
| *name*.length() | a.length() |