SHARK

Contents

VARIABLES

In SHARK, we can initial a variable by using VAR keyword:

```
SHARK > VAR name = “Hamed”

Hamed

SHARK > name

Hamed

SHARK > VAR age = 20

20

SHARK > age

20
```

COMPARISONS OPERATORS

For comparing, we must use comparison operators. Comparison operators are: < , > , >= , <= , == , !=

In output, 1 means True and 0 means False.

```
SHARK > 5 == 5

1

SHARK > 10 >= 12

0

SHARK > 4 < 6

1

SHARK > 8 == 7

0

SHARK > 6 != 8

1
```

LOGICAL OPERATORS

For implement logical statements, we must use logical operators. Logical operators are: AND, OR

In output, 1 means True and 0 means False.

```
SHARK > 5 == 5 AND 6 >= 4

1

SHARK > 10 >= 12 AND 8 == 8

0

SHARK > 4 < 6 OR 8 > 6

1

SHARK > 8 == 7 OR 6 == 5

0

SHARK > 6 != 8 OR 1 != 1

1
```

IF STATEMENTS

The if statement is used for conditional execution. We set our condition after IF and its expression after THEN. Also, we can set related conditions by using ELIF and ELSE.

NOTE: SHARK supports single-line IF statement.

```
SHARK > VAR number = 25

1899

SHARK > IF number > 0 THEN "Positive" ELIF x < 0 THEN "Negative" ELSE
"Zero"

Positive
```

Initial a variable depending on result of IF statement:

```
SHARK > VAR age = 19

1

SHARK > VAR price = IF age > 18 THEN 40 ELSE 20

40

SHARK > price

40
```

WHILE STATEMENTS

The while statement is used for repeated execution as long as an expression is true:

NOTE: You have to use VAR after THEN if you want to use an initialized variable.

```
SHARK > VAR i = 0

0

SHARK > WHILE i < 1000 THEN VAR i = i + 1

SHARK > i

1000
```

FOR STATEMENTS

A for loop repeats until a specified condition evaluates to false. For iterating over a sequence of numbers we set limitation by TO. In multi-line FOR statements, you must end the loop by END keyword.

NOTE: You have to use VAR if you want to use an initialized variable in expression.

```
SHARK > VAR result = 1

1

SHARK > FOR i = 1 TO 6 VAR result = result * i

SHARK > result

120
```

Multi-line FOR statement:

```
FOR i = 0 TO 6 THEN

      VAR result = result * i

END
```

BREAK AND CONTINUE STATEMENTS

The BREAK statement, breaks out of the innermost enclosing FOR or WHILE loop.

Loop statements may have an else clause; it is executed when the loop terminates through exhaustion of the iterable (with FOR) or when the condition becomes false (with WHILE), but not when the loop is terminated by a BREAK statement.

```
SHARK > VAR numbers = []

1

SHARK > FOR i = 1 TO 12 THEN; IF i == 5 THEN CONTINUE ELIF i == 9 THEN
BREAK; VAR numbers = numbers + i; END

SHARK > numbers

1, 2, 3, 4, 6, 7, 8
```

DEFINE FUNCTIONS

The keyword DEF introduces a function definition. It must be followed by the function name and the parenthesized list of formal parameters. The statements that form the body of the function start at the next line, and when body function ends, we must use END keyword.

```
DEF join(elements, len, separator)
      VAR result = ""

      FOR i = 0 TO len THEN
            VAR result = result + elements/i
            IF i != len - 1 THEN VAR result = result + separator
      END

      RETURN result
END
```

NOTE: Also, you can some specific functions like this:

```
DEF sharkify(prefix) -> prefix + " Shark"
```

`sharkify` function returns the prefix by adding the " Shark" suffix.

The `->` defines the RETURN keyword in this function.

STRINGS

We can initial a string by wrap our characters in “”.

```
SHARK > “SHARK is a programming language”

SHARK is a programming language
```

Also, for concatenation, we should use +.

```
SHARK > “SHARK is ” + “a programming language”

SHARK is a programming language
```

Then, we can multiply strings by *.

```
SHARK > “Baby SHARK, doo-doo, doo-doo ” * 3 + “Baby SHARK”

Baby SHARK, doo-doo, doo-doo Baby SHARK, doo-doo, doo-doo Baby SHARK, doo-
doo, doo-doo Baby SHARK
```

LISTS

SHARK knows a number of compound data types, used to group together other values. The most versatile is the list, which can be written as a list of comma-separated values (items) between square brackets. Lists might contain items of different types, but usually the items all have the same type.

For appending to list, we use +.

```
SHARK > [1, 2, 3] + 4
[1, 2, 3, 4]
```

For concatenate two lists, we use *.

```
SHARK > [1, 2, 3] * [4, 5, 6]
[1, 2, 3, 4, 5, 6]
```

For returning element of a specific index, we must use index after /.

```
SHARK > [1, 2, 3] / 1
2
```

BUILT-IN FUNCTIONS

Any function that is provided as part of a high-level language and can be executed by a simple reference with specification of arguments is a built-in function.

SHARK has some built-in functions:

> PRINT()

PRINT() function prints its input:

```
SHARK > PRINT(“Hamed Khosravi”)

Hamed Khosravi
```

> INPUT()

INPUT() function prompts the user for data that is converted to and returned as a string.

```
SHARK > VAR name = INPUT()

Hamed

SHARK > “My name is ” + name

My name is Hamed
```

> INPUT_INT()

INPUT_INT() function prompts the user for data that is converted to and returned as a integer.

```
SHARK > VAR birthdate = INPUT_INT()

2002

SHARK > 2023 – birthdate

21
```

> CLEAR() / CLS()

If your OS is Windows or MacOS you can clear the screen by CLS() and If you are a Linux lover you must use CLEAR().

> IS_NUM()

IS_NUM() function checks if its argument is number data type. Returns 1, If it's number, else 0.

```
SHARK > IS_NUM(21)

1

SHARK > IS_NUM(“Hamed”)

0
```

> IS_STR()

IS_STR() function checks if its argument is string data type. Returns 1, If it's string, else 0.

```
SHARK > IS_STR(“Hamed”)

1

SHARK > IS_STR(21)

0
```

> IS_LIST()

IS_LIST() function checks if its argument is list data type. Returns 1, If it's list, else 0.

```
SHARK > IS_LIST([1, 2, 3])

1
```

> IS_DEF()

IS_DEF() function checks if its argument is function data type. Returns 1, If it's function, else 0.

```
SHARK > IS_DEF(PRINT)

1
```

> APPEND()

APPEND() function appends the second argument to the first argument.

NOTE: The first argument must be list.

```
SHARK > VAR list = [1, 2, 3]

1, 2, 3

SHARK > APPEND(list, 4)

SHARK > list

1, 2, 3, 4
```

> POP()

POP() functions takes two arguments. The first one is a list and the second one is the index of the elements which is going to be popped.

```
SHARK > VAR list = [1, 2, 3, 4]

1, 2, 3

SHARK > POP(list, 2)

3
```

> EXTEND()

EXTEND() functions takes two lists ad arguments. This function concatate the second argument to the first.

```
SHARK > VAR list = [1, 2, 3, 4]

1, 2, 3

SHARK > EXTEND(list, [5, 6, 7, 8])

SHARK > list

1, 2, 3, 4, 5, 6, 7, 8
```

> LEN()

LEN() function returns the length of a list.

```
SHARK > VAR list = [1, 2, 3, 4, 5, 6, 7, 8]

1, 2, 3, 4, 5, 6, 7, 8

SHARK > LEN(list)

8
```

> RUN()

RUN() function takes a shark file as an argument and executes it.

`main.shark`

```
PRINT(“Hello, World!”)
```

```
SHARK > RUN(“main.shark”)

Hello, World!”
```

COMMENTS

a comment is a programmer-readable explanation or annotation in the source code of a computer program. They are added with the purpose of making the source code easier for humans to understand, and are generally ignored by compilers and interpreters. SHARK supports comments in executable files (.shark).

main.shark

```
# This program prints the “Hello, World!” and developed by Hamed Khosravi.

PRINT(“Hello, World!”)
```

```
SHARK > RUN(“main.shark”)

Hello, World!”
```