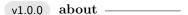# 1 Standard Library Functions

about ————————

It is mandatory for all Anterogradia libraries to implement an *about* function. The standard library is no exception to this rule. The standard implementation provides basic information about the lib at hand.

```
about ()
```

sequence ————————

The **variadic** *sequence* function evaluates every parameter expression and returns a string made up of all individual results.

```
sequence {
    "Hello, "
    "World!"
}
    => "Hello, World!"
```

progn ————————

Just like in Common Lisp, the **variadic** *progn* function evaluates all parameters in sequence and returns the last value.

```
progn {
    "Hello, "
    "World!"
}
    => "World!"
```

nothing ————————

Returns an empty string (a string with length 0).

```
nothing ()
    => ""
```

repeat ————————

Repeats the expression *str* for *count* times, each iteration separated by an optional *separator*, otherwise unseparated.

```
repeat (count = 3, str = "Hello")
    => "HelloHelloHello"

repeat (count = 3, str = "Hello",
        separator = " ")
    => "Hello Hello Hello"
```

random ————————

The **variadic** function *repeat* randomly evaluates a single expression.

```
random {
    "Foo"
    "Bar"
    "Baz"
}
    (1) => "Bar"
    (2) => "Baz"
    (2) => "Baz"
    (3) => "Bar"
    (4) => "Foo"
    ...
```

_if ————————

The *_if* function implements conditional control flow. If *cond* is `"true"`, the *then* expression will be evaluated and returned as the result of the function. Otherwise, the function evaluates and returns the *else* expression.

```
_if (cond = 1 > 2,
    then = "1 is bigger than 2!",
    else = "Math still works!")

    => "Math still works!"
```

equal ————————

The *equal* function compares the expressions *left* and *right*. If both have the same value, the function returns `"true"`, otherwise it returns `"false"`

```
equal (left = "123", right = "321")
    => "false"
```

param ————————

Anterogradia may be started with custom startup parameters from within the Kotlin API. This function is used to retrieve said parameters, with *key* being the key of a given startup parameter entry.

```
param (key = "binaryPath")
```

set ————————

This function together with *get* implement the backbone of Anterogradia's memory features. This function creates or modifies a variable identified by a *key* with a given *value*. This function always returns an empty string.

```
set (key = "message", value = "Hello, World!")
    => ""
```

v1.0.0 **get** ———————

The *get* function retrieves a variable *key* and returns the value.

```
get (key = "message")
    => "Hello, World!"
```

v1.0.0 **compile** ———————

Thsi function is used to dynamically invoke the Anterogradia interpreter while re-using the current runtime object. Thus, all libraries, functions and variables present in the host script are going to be usable in the code passed to the *source* parameter.

```
progn {
    set (key = "msg", value = "Hi!")
    compile (
        source = "&`msg"
    )
}
    => "Hi!"
```

v1.0.0 **lgt** ———————

The *lgt* function compares the expressions *left* and *right* and returns `"true"` if the former is greater than the latter; otherwise `"false"` Depending on the value of both expressions, the comparison will either be numeric or lexicographic.

```
lgt (left = "123", right = "321")
    => "false"
```

v1.0.0 **rgt** ———————

Same as *lgt*, but `(right > left) ? "true" : "false"`

```
rgt (left = "123", right = "321")
    => "true"
```

v1.0.0 **len** ———————

Returns the length of the *expr* string.

```
len (expr = "Hello!")
    => "6"
```

v1.0.0 **astd** ———————

Generates valid Anterogradia source code from the parser result of the passed *expr*.

```
astd (expr = get(key = "abc"))
    => "get(key="abc")"
```

v1.0.0 **__fun** ———————

This function stores the *expr* expression as *id*. It is worth mentioning that, unlike variables, what gets stored is not the result of evaluating the given expression, but rather the AST nodes making up said expression. Thus, evaluating such stored expressions **might** yield different values on each iteration.

```
_fun (id = "greet", expr = &`abc)
```

v1.0.0 **__eval** ———————

The **__eval** function is closely related with the **__fun** function. Its purpose is to retrieve the expression *id* stored via the former function and evaluate it at a given point in time.

```
sequence {
    set(key = "abc", value = "Hi!")
    _eval (id = "greet")
    " "
    set(key = "abc", value = "Hello!")
    _eval (id = "greet")
}
    => "Hi! Hello!"
```

v1.0.0 **___require_prop** ———————

This function checks for the existence of the variable *id* and causes the interpreter to throw an `AntgRuntimeException` with the *err* message whenever it cannot find the required variable. Note that the existence of a variable is determined by the value of its length being `> 0`

```
__require_prop(
    id = "abc",
    err = "Variable not found!")
```

```
(This causes the runtime to throw the
aforementioned exception, since the
variable is not present in this context.
This also means that the execution of
the script will be interrupted at
this exact point.)
```

It is worth mentioning, that this function is a utility designed to implement reliable function calls and was originally meant to be generated exclusively by the **function definition syntax binding**. It is not recommended to use it manually.

v1.0.0 **add** ———————

Evaluates to the result of adding the *left* and *right* operands together.

```
add (left = "10", right = "2")
    => "12"
```

v1.0.0 **sub** ———————

Evaluates to the result of subtracting the *right* operand from the *left* operand.

```
sub (left = "10", right = "2")
    => "8"
```

v1.0.0 **mul** ———————

Evaluates to the result of multiplying the *left* and *right* operands together.

```
mul (left = "10", right = "2")
    => "20"
```

v1.0.0   **div** ─────────

Evaluates to the result of dividing the *left* operand by the *right* operand.

```
div (left = "10", right = "2")
    => "5"
```

v1.0.0   **mod** ─────────

Evaluates to the result of retrieving the division remainder of *left / right*.

```
mod (left = "10", right = "2")
    => "0"
```

v1.0.0   **signflp** ─────────

Evaluates to *expr* with a flipped sign.

```
signflp (expr = "123")
    => "-123"
```

v1.0.0   **vsignflp** ─────────

Performs a sign-flip on the variable *id* and stores the result in the source variable.

```
progn {
    set (key = "a", value = "12")
    vsignflp (key = "a")
    &`a
}
    => "-12"
```

v1.0.0   **increment** ─────────

Increments the value of the variable *id* and stores the result in the source variable.

```
progn {
    set (key = "a", value = "10")
    increment (id = "a")
    &`a
}
    => "11"
```

v1.0.0   **decrement** ─────────

Decrements the value of the variable *id* and stores the result in the source variable.

```
progn {
    set (key = "a", value = "10")
    decrement (id = "a")
    &`a
}
    => "9"
```