Synt Documentation

Release latest

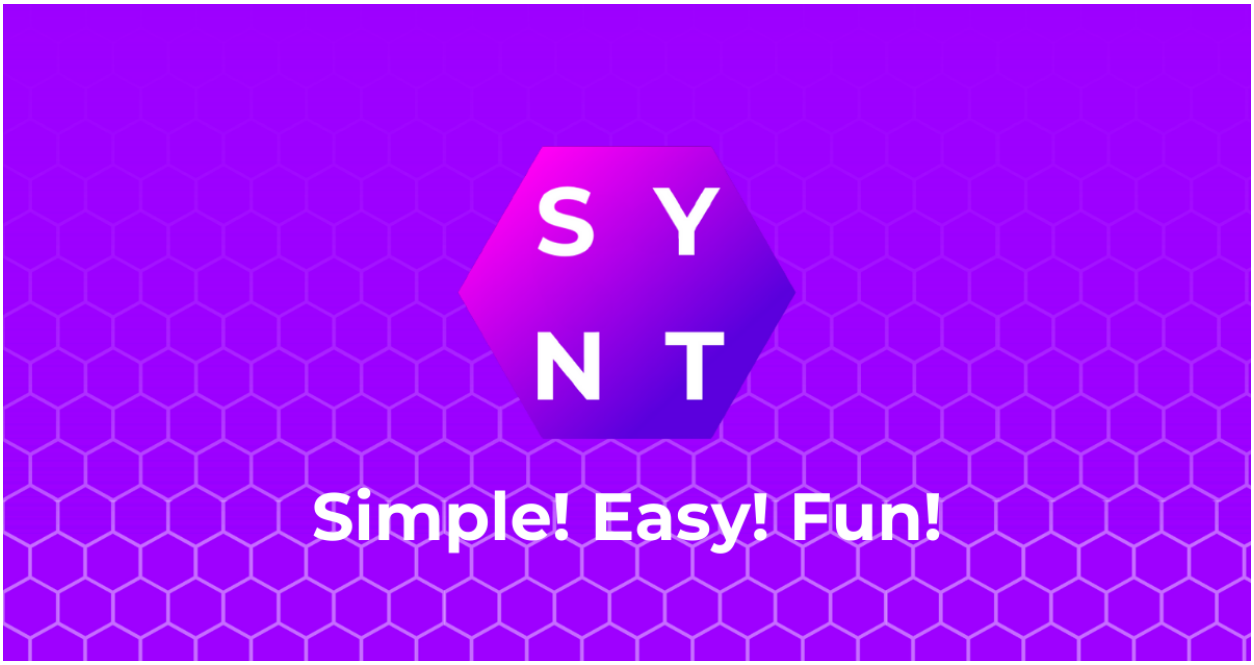
Jun 02, 2022

CONTENTS

1 Table of Contents **3**

2 General **7**
- 2.1 What is Synt? . . . 7
- 2.2 Examples . . . 7
- 2.3 Installation . . . 8
- 2.4 Modes . . . 8

3 Learn Synt **11**
- 3.1 What is Console? . . . 11
- 3.2 How Synt Reads Code? . . . 11
- 3.3 Commenting in Synt . . . 11
- 3.4 Basic Algorithms of Synt . . . 12
- 3.5 Dynamic Programming in Synt with Variables . . . 13
- 3.6 Avoiding Redundancy in Synt with Functions . . . 14
- 3.7 Synt Mathematical Operations . . . 15
- 3.8 Conditional Programming with Synt . . . 16
- 3.9 Special Characters . . . 17
- 3.10 Repitition and Looping Code . . . 18
- 3.11 Functions for Collections and Other Iterables . . . 18
- 3.12 Reading and Writing Files . . . 19
- 3.13 Synt Time and Ticking . . . 20
- 3.14 Handling Running Console with Synt . . . 21
- 3.15 Debugging in Synt . . . 21
- 3.16 Modules and External Resources . . . 22


S Y
N T
Simple! Easy! Fun!

CHAPTER

ONE

TABLE OF CONTENTS

1. General
    1. What is Synt?
        - Examples
    2. Installation
        - Download
        - Move to Path
        - Add to Path
    3. Modes
        - File
        - Interactive
        - Compile
        - Quit
        - SPyC
        - SPyC Run
2. Learn Synt
    1. Console Basics
    2. Code Iterations
    3. Commenting
        - General
        - Info
        - Command
    4. Basic
        - Version
        - End
        - Output
        - Input
    5. Variables

- Meaning
- Types
- Creation
- Using Collection Items

6. Functions
   - Meaning
   - Creation
   - Using
   - Call Syntax
   - Return Values
7. Operator
   - Add
   - Subtract
   - Multiply
   - Divide
   - Power
8. Logic
   - Check
   - Condition
   - Operators
9. Special Characters
10. Loop
    - Repeat
    - Loop
11. Iterable Functions
    - Count
    - Insert
    - Remove
    - Delete
12. File
    - Read
    - Write
13. Time
    - Ticks
    - Time
    - Reset Tick

- Pause Tick
- Resume Tick
- Get Tick

14. Console
    - Console
    - Clear
15. Debug
    - Variable Type
    - Restore
    - Error
    - Warn
16. Modules
    - Importing

CHAPTER

TWO

GENERAL

2.1 What is Synt?

Synt is a programming language and it is very suitable for beginners. Code is easy and really simple. Syntax is user friendly are really readable for person who can read and understand English. Synt is free and open source. It is made with Python. Synt is the next gen programming language. It is made so that updates are easy and fast. Synt is made to make software and game development easier and faster. For the history of Synt, it was initiated by `Attachment Aditya <https://attaditya.is-a.dev/>`__ in year 2020. It was initially expected to be worked for about 1-2 Months. Later as time went by, it was expanded to be a larger project. It will most probably be getting updated and expanded in the future.

2.2 Examples

2.2.1 Hello Synt

```
output "Hello Synt!"
```

2.2.2 Create Variables

```
var number my_num1 0
```

2.2.3 Add Numbers

```
var number my_sum 0
add my_sum 10 20 30 40 50 60
```

2.2.4 Variable Referencing

```
var text name "Synt"
output "Hello #name#!!! Welcome Back!!!"
```

2.3 Installation

Synt is created in a way so that it can be run in any systems. However, the major focus is for `Windows`. For other system Synt has `Python Source Code` which can be run to use Synt. In future, there might be versions of Synt targetted for other platforms. This section is focussing to install synt on `Windows`. For other platforms you need to have `Python` installed on your system. After that install the `Source Code` of Synt from `GitHub`. Run the `Python Source Code` using `Python` to use Synt.

2.3.1 Download Synt

Download `Synt Executable` for you through `Official Synt Site <https://synt.ml/#downloads>`__.

2.3.2 Move to Path

Installing Synt just means moving it to the `Path` you want it to be. It can be done manually. This allows Synt to be used portably. Generally, for `Windows`, it is recommended for `Apps` to be installed in `Programs Files`. Although this is not mandatory, it is on the user where to install it.

```
Suggested Synt Folder: `C:\Program Files\Synt`
```

2.3.3 Add to Path

Although you might have installed synt in the `Path` you want it to be in, but you still need to access it. To access synt, you may need to use the `Path` again and again to refer to the `Synt Executable` that will run Synt Code. A simple solution is to add the `Path` to your `Path` in your `Environment Variables`. This might sound complicated, but it isn't really that hard. First of all, copy the `Folder Path` you want to add to your `Path` to your `Environment Variables`. Now, open your `Environment Variables`. To do so, start `Run`(`Windows Key` + `R`). In run, type `sysdm.cpl`. Once this causes the `System Properties` to open, click on `Advanced`. Now, click on `Environment Variables`. Under `System Variables`, click on `Path` and click on `Edit`. Now, click on `New` and then paste the `Folder Path`. Finally, click on `Apply` and `OK` on all windows till all `System Properties` windows are closed. Synt is now globally added to `Path`. You can just refer `Synt` to run Synt.

2.4 Modes

On using Synt without console arguments, it asks you for modes. Synt has two major running modes. `File Mode` and `Interactive Mode`. Apart from modes that can run code, there are more modes. `Quit Mode` and `Compile Mode` are two of them. Synt also allows code to convert into SPyC (SyntPythonCompiler) that can be run with `Python`.

2.4.1 File Mode

To run an existing code using `File Mode` is suggested. This mode will ask you file path. It is the complete file location to your code file. Once code completed it closes. This mode can be also accesed from directly passing `source code file path` in command line arguments. The mode key is `*f` and `*file`.

```
synt *f
```

```
synt *file
```

```
synt "main.synt"
```

2.4.2 Interactive Mode

`Interactive Mode` or `Synteractive` is a good mode to learn and test commands. This mode allows you to type and run code at same time. As it runs at same time, it doesn't close until closing through code or force close. The mode key is `*i` and `*interactive`.

```
synt *i
```

```
synt *interactive
```

2.4.3 Compile Mode

`Compile Mode` is a `command line arguments` based mode. This mode `compiles` from `synt code` to an `distributable` application. The mode key is `*c` and `*compile`.

```
synt *c "main.synt"
```

```
synt *compile "main.synt"
```

2.4.4 Quit Mode

`Quit Mode` just closes Synt. The mode key is `*q` and `*quit`.

```
synt *q
```

```
synt *quit
```

2.4.5 SPyC Mode

`SPyC Mode` is a `command line arguments` based mode. This mode compiles from `synt code` to `Python Source Code`. The mode key is `*spyc`.

```
synt *spyc "main.synt"
```

2.4.6 SPyC Run Mode

`SPyC Run Mode` is a `command line arguments` based mode. This mode directly exports and runs the `SPyC Output`. The mode key is `**spyc`.

```
synt **spyc "main.synt"
```

CHAPTER

THREE

LEARN SYNT

3.1 What is Console?

The console is the main window that opens when Synt or Synt Code is run. It is the place where you can type and run code in Interactive Mode. In file mode you can use console to see the outputs of code and give code inputs.

3.2 How Synt Reads Code?

First, obviously, Synt gets the code it needs to run. This code can also be called as `source code`. Then, Synt breaks the code into `blocks`. The `blocks` into `tokens`. Synt uses these blocks and tokens to recognize and execute corresponding commands and code. Synt runs `block after block`.

3.3 Commenting in Synt

There are three `Commenting Character` options in synt. The functioning and working of all three is same. All of them will be ignored during execution. These are classified on the basis of organizing and how they are supposed to be used.

3.3.1 General Comments

Such comment is an algorithm in synt that will be ignored. It has no use except for being used as placeholders and of course to comment in between code.

```
comment This is a comment. This will be ignored in execution.
```

```
$ This is a comment. This will be ignored in execution.
```

3.3.2 Info Comments

This type of comment is used to `organize` code. This should be generally used to convey what the following code is about and what it should do.

```
? This is a query comment. This will be ignored in execution.
```

3.3.3 Command Comments

This type of comment is used when using an `AI` to write or analyze code. Its content will depend on the `AI's` and `custom mods` that are used.

```
> This is an AI command comment. If any AI or mod installed, this might do something.
> It won't do anything in execution.
```

3.4 Basic Algorithms of Synt

There are a few basic functions or so called `Algorithms` in synt. These include basic `input` and `output` functions, as well as `version` function and `end` function. These are the most primitive functions in synt.

3.4.1 Version

This function just outputs the `version` of synt in console.

```
version
```

3.4.2 End

This function just ends/pauses the execution of synt.

```
end
```

3.4.3 Output

This function outputs the `arguments` passed into the function to the console.

```
output "Hello Synt!"
```

3.4.4 Input

This function is slightly more complex than the `output` function. The first argument it takes is the `Output Variable`. This is the variable that the `Input` will be stored in. The second argument is the `Input Prompt`. This is basically the text that should output before user is asked to input something.

```
input input_var "Enter your input:"
```

3.5 Dynamic Programming in Synt with Variables

3.5.1 What is a Variable in Synt?

Synt allows your program to be more `dynamic` with the help of `variables`. `Variables` are a `sets of characters` that contain some `value` inside them. It can also be said as `naming` some value and then using the name instead of value later on.

3.5.2 Different Types of Variables

Synt offers a few types of basic `variables`. These include `number`, `decimal`, `text`, `binary`, `collection` and `nothing`. A `number` is a `number` that can be `positive`, `negative` or `zero`. It can't contain anything apart from `numeric characters` and `negative sign`. A `decimal` is just like a `number type` that can contain `decimal point`. A `text` is a `set of characters` enclosed between `double quotes(")`. A `binary` is a `number` that can only contain `on` and `off`. It can also be considered as `TRUE` or `FALSE` values, `0` or `1` and `empty` or `non-empty` values. A `collection` is a `set of items` enclosed between `square brackets([])` and separated by `new line(\n)`. These can contain other types inside themselves. A `nothing` is a `variable` that has no value.

```
number ..., -3, -2, -1, 0, 1, 2, 3, ...
decimal ..., -3.5, -2.5, -1.5, 0.5, 1.5, 2.5, 3.5, ...
text "hello", "yo", "SYNT!!!!", "SYYYYYYYYYYYYYYYYYYYYNT!!!!", ...
binary on, off
nothing
collection [
    "Text Type"
    "Another Text Type"
    10
    -10
    0
    0.72
    -55.22
    on
    off
]
```

3.5.3 Making a Variable

To make a variable, you can use the `var` function.

```
var type var_name "var_value!"
```

3.5.4 Using a Variable

To use a variable, you can use the `variable name` enclosed between `hash(#)`.

```
output "Use a variable like #var_name#"
```

3.5.5 Using an Item in a Collection

To get an item from a collection, first refer collection as a variable and then mention the `item index number` enclosed between < > just immediately after the collection variable reference. `item index number` is the amount of items in a collection before the item you want to get. In simple terms the `item index number` is the `position` of the item in the collection minus 1.

```
output "Lets say collection coll_name has item #coll_name#<0> and #coll_name#<1>!"
```

3.6 Avoiding Redundancy in Synt with Functions

Repitition of code is a `bad practice` and can make your code cost more time to execute. Synt however offers the ability to `avoid` this by using `functions`.

3.6.1 Declaring a Function

A `function` is a `set of instructions` that can be called/referenced to execute the `instructions` inside it for which it was defined.

3.6.2 Making a Function

`custom functions` are defined by using the `alg` function. They take name of list that will contain arguments which can later be used inside the instructions and the function name as parameters. The `instructions` are enclosed between `{ }`.

```
alg func_name argument_collection {
    output "Created a function. Calling first argument of the function. #argument_
↪collection#<0>!"
}
```

3.6.3 Using a Function

After the function is defined it can be called simply by using the `function name` at the start of the line like other functions.

```
func_name
```

3.6.4 Proper Syntax for a calling a custom Function in Synt

`custom functions` can be called in Synt easily but they also take more parameters like the `return value variable` and the `arguments` passed.

```
func_name return_variable arg1 arg2 ...
```

3.6.5 Returning a Value from a Function

To return a value from a function, use the `result` function.

```
alg func_name argument_collection {
    output "Created a function. Calling first argument of the function. #argument_
↪collection#<0>!"
    result "Returned value!"
}
```

3.7 Synt Mathematical Operations

Synt allows user to perform some simple and complex mathematical operations on numbers, decimals, texts and other types.

3.7.1 Adding Numbers

To add two numbers, use the `add` function.

```
add output_variable 2 3
? this sets output_variable to 5
```

3.7.2 Subtracting Numbers

To subtract two numbers, use the `subtract` function.

```
subtract output_variable 2 3
? this sets output_variable to -1
```

3.7.3 Multiplying Numbers

To multiply two numbers, use the `multiply` function.

```
multiply output_variable 2 3
? this sets output_variable to 6
```

3.7.4 Dividing Numbers

To divide two numbers, use the `divide` function.

```
divide output_variable 6 3
? this sets output_variable to 2
```

3.7.5 Power

To get the power of a number, use the `power` function.

```
power output_variable 2 3
? this sets output_variable to 8
```

3.8 Conditional Programming with Synt

In many cases, you may want to perform some action based on some condition. Synt provides a way to do this with conditional programming. Synt has two functions for these cases. The first one can be used to check if a condition is true or false. The second one can be used to perform an action if the condition is true or false.

3.8.1 Check and Return

To check if a condition is true or false, use the `check` function.

```
check output_var compare_var_1 check_operation compare_var_2
? this sets output_var to on if the condition is true, and to off if the condition is␣
↪false
```

3.8.2 Check and Perform

To perform an action if a condition is true or false, use the `condition` function.

```
condition compare_var_1 check_operation compare_var_2 {
    output "Condition is true!"
} {
    output "Condition is false!"
}
? this will output "Condition is true!" if the condition is true else "Condition is␣
↪false!"
```

```
condition compare_var_1 check_operation compare_var_2 true_action false_action
? this will execute the true_action if the condition is true else the false_action
```

3.8.3 All Conditional Operations

Here's the list of all the conditional operations that can be used with `check` and `conditional` functions:

```
"equals to"               equals        =
"not equals to"           not           !=
"greater than"            greater       >
"less than"               less          <
"greater than equal to"   notless       >=
"less than equal to"      notgreater    <=
"contains"                contains      <-
"does not contain"        notcontains   !<-
"starts with"             starts        _%
"does not start with"     notstarts     !_%
"ends with"               ends          %_
"does not end with"       notends       !%_
```

3.9 Special Characters

Sometimes a character might be needed in a text but it can't be used directly. In such cases special character keywords are used which will replace the keyword with the special character. Here's the list of special characters that can be used in Synt:

```
#NEWLINE        '\n'
#INDENT         '\t'
#BACKSPACE      '\b'
#START          '\r'
#SPACE          ' '
#LEFTSQUARE     '['
#RIGHTSQUARE    ']'
#LEFTCURLY      '{'
#RIGHTCURLY     '}'
#COMMA          ','
#DOT            '.'
#SEMICOLON      ';'
#COLON          ':'
#EQUAL          '='
#HASH           '#'
#QUESTION       '?'
#EXCLAMATION    '!'
#QUOTE          '"'
#APOSTROPHE     '''
```

3.10 Repitition and Looping Code

When some code or function is needed to repeated `simultaneously` with some or no variation then the writing everything `manually` is really difficult. And when repitition to be done according to a variable or till the condition is true, that is dynamically changing to a variable, then the writing is even more difficult. To make the writing of such code easier, Synt also allows another common feature in programming languages called `looping`. `Loops` are used to repeat a block of code a number of times. In Synt, there are two types of loops. One is `statically repeating` and the other is `dynamically repeating`. Loops that are `statically repeating` just follow a given number of times and then stop. These don't update the argument dynamically and are just a method to reduce `code redundancy`. This type of loop can be used with the `repeat` function. Loops that are `dynamically repeating` are used to repeat a block of code a number of times that isn't fixed. These are used to update the argument dynamically. This type of loop can be used with the `loop` function.

3.10.1 Statically Repeating Loops

The `repeat` function is used to repeat a block of code a given number of times.

```
alg func args {
    output "Hello Synt!"
}
repeat 10 func
? executes func 10 times
```

3.10.2 Dynamically Repeating Loops

The `loop` function is used to repeat a block of code a number of times that is dynamically changing.

```
? count down
var number cd 10
var binary do_it

alg func args {
    output #cd#
    subtract cd #cd# 1
    check do_it cd >= 0
}

loop do_it func
? count downs to 0
```

3.11 Functions for Collections and Other Iterables

Iterables are the variable types that are made up of multiple values. These include `text` composed of `characters` and `collections` containing `items`. Sometimes you need to use an iterable to store multiple values and data. In such cases you would need to get length, add, remove and do other stuff with the iterable. Synt allows such functions.

3.11.1 Length of an Iterable

To get the `length` of an `iterable`, you can use the `count` function.

```
var number len
count len obj
? returns the length of obj, obj is predefined
```

3.11.2 Inserting an Item to an Iterable

To insert an `item` into an `iterable`, you can use the `insert` function.

```
insert obj item index
? inserts item into obj at index, obj is predefined
```

3.11.3 Removing an Item from an Iterable

To remove an `item` from an `iterable`, you can use the `remove` function.

```
remove obj item limit
? removes item from obj, obj is predefined, removes limit amount of item
```

3.11.4 Removing an Item at Given Index from an Iterable

To remove an `item` at a given `index` from an `iterable`, you can use the `delete` function.

```
delete obj index
? removes item at index from obj, obj is predefined
```

3.12 Reading and Writing Files

Synt allows you to access local files on your computer. This could be helpful to create cache for next run or to save data. Synt can read and write files.

3.12.1 Reading a File

To read a file, you can use the `read` function. Just note that the file should exists.

```
var text data
read data "file.txt"
? reads "file.txt" and stores it in data
```

3.12.2 Writing a File

To write a file, you can use the `write` function. If file does not exist, it will be created. If file does exist, it will be overwritten.

```
write "file.txt" data
? writes data to "file.txt"
```

3.13 Synt Time and Ticking

3.13.1 Ticks in Synt

Synt has a customizable `tick system` that allows using and calculating time slightly easier. One tick is equal to one millisecond. Ticks can be reset, paused and resumed. It can also be set to a specific value.

3.13.2 Time in Synt

Synt also has a `time system`. This system can not take any inputs. It is made only for outputs. It isn't affected by ticks.

3.13.3 Resetting Ticks

To reset ticks, you can use the `reset_tick` function. This will set the tick value to 0, which was initial value.

```
reset_tick
? resets ticks
```

3.13.4 Pausing Ticks

To pause ticks, you can use the `pause_tick` function. This will stop updating ticks until resumed.

```
pause_tick
? pauses ticks
```

3.13.5 Resuming Ticks

To resume ticks, you can use the `resume_tick` function. This will resume updating ticks. This might give you an error if you try to resume ticks when they are already resumed.

```
resume_tick
? resumes ticks
```

3.13.6 Getting Ticks

To get ticks, you can use the `get_tick` function. This will return the current tick value.

```
var number this_tick
get_tick this_tick
? returns ticks and stores it in this_tick
```

3.14 Handling Running Console with Synt

When Synt is unable to do something directly, you might think to access console to run it through other means. This is possible because Synt has the ability to access the console it is being run on. You can also use console commands to change the other console properties.

3.14.1 Run Console Commands

Synt has a `console` function that allows you to run console commands. This is useful for changing the console properties that you want to but can't directly using Synt. Not only console properties but if you want even run other console commands, you can use this function.

```
console "command"
? runs "command" in console
```

3.14.2 Clearing Console

To clear the console, you can use the `clear` function. This will clear the console.

```
clear
? clears console
```

3.15 Debugging in Synt

Synt has an useful set of `debug functions` that can be used to debug your code. Debugging means to find out what is happening in your code. Generally it is used to find out what is wrong with your code, that is wy you are getting an error. However, it can also be used to understand how your code works.

3.15.1 Get Variable Type

To get the type of a variable, you can use the `info` function. This will return the type of the variable.

```
var number this_number
var text type
info type this_number
? returns type of this_number and stores it in type
```

3.15.2 Continuing Program after Error

To continue the program after an error, you can use the `restore` function. This will continue the program after an error.

```
restore
? continues program after error
```

3.15.3 Custom Error Messages

To create a custom error message, you can use the `error` function. This will create and run a custom error message.

```
error "message"
? displays error with "message" as content
```

3.15.4 Custom Warning Messages

To create a custom warning message, you can use the `warn` function. This will create and run a custom warning message.

```
warn "message"
? displays warning with "message" as content
```

3.16 Modules and External Resources

Sometimes you want to organize your code in modules or other files that need to be run. Synt allows you to import and run modules. When you import a module, it will run the code in the module. If there are any custom functions in the module, they will be available in the main file and while running in other modules that will be imported orderwise next.

3.16.1 Importing Local Files as Modules

To import a local file as a module, you can use the `module` function. This will import and run the module.

```
module "module_name"
? imports file "module_name.synt" as module and runs it
```

Synt