Programming in Suny

1. Introduction

Suny is a lightweight scripting language designed to be **simple, readable, and beginner-friendly**. It is ideal for people who are just starting programming, but also powerful enough to solve real problems.

Suny is written in **C**, which makes it **fast, efficient, and portable** across different platforms. Its syntax is clean, minimizing boilerplate code and focusing on **clarity and simplicity**.

Key Features

- **Simplicity:** Minimal and easy-to-remember syntax.
- **Clarity:** Readable code that is easy to understand.
- **Flexibility:** Supports a wide range of programming tasks without complexity.

Typical Uses

- Learn programming concepts quickly.
- Write small scripts or automation tasks.
- Experiment with application development in a lightweight way.

Suny encourages **learning by doing**, allowing users to experiment interactively and receive instant feedback through its REPL.

---

2. Getting Started

All programs involve **input** (getting data from the user) and **output** (displaying results):

- **Input:** Receives information from the user (usually as a string).
- **Output:** Displays text, numbers, or other information on the screen.

Example: Printing a Message

```
print("Hello, Suny!")
```

Output:

```
Hello, Suny!
```

Explanation:

- `print()` is used to display text or values.
- Text inside double quotes `" "` is a **string**, representing letters, numbers, or symbols.

Running Suny Programs

1. Save your code in a file with the extension `.suny`.
2. Run it using the Suny interpreter:

```
prompt> suny my_program.suny
```

Example:

```
prompt> suny main.suny
Hello, Suny!
prompt>
```

Using the REPL

Suny includes a **REPL** (Read-Eval-Print-Loop) for interactive programming:

```
prompt> suny
Suny 1.0 Copyright (C) 2025-present, by dinhsonhai132
>> print("Hello, REPL!")
Hello, REPL!
>>
```

How the REPL Works:

1. **Read:** Reads your input code.
2. **Eval:** Executes the code.
3. **Print:** Displays the result.
4. **Loop:** Repeats the process until you exit.

Advantages of REPL:

- Instant feedback while learning.
- Test small code snippets without creating files.
- Experiment safely without affecting saved code.

---

3. Simple Math and Operators

Suny supports **basic arithmetic** and **comparison operations**, allowing you to perform calculations and make decisions.

Arithmetic Operations

```
print(2 + 3)       # Addition: 5
print(5 - 2)       # Subtraction: 3
print(4 * 2)       # Multiplication: 8
print(10 / 2)      # Division: 5.0
print((1 + 2) * 3) # Parentheses control order: 9
```

Comparison Operators

Comparison operators are used to compare values. They return **Boolean values** (`true` or `false`):

```
# Comparison examples
print(3 < 5)   # true
print(5 > 3)   # true
print(2 == 2)  # true
print(2 <= 3)  # true
print(5 >= 5)  # true
```

Explanation:

- `<` → less than
- `>` → greater than
- `==` → equal to
- `<=` → less than or equal to
- `>=` → greater than or equal to

---

4. Variables

Global Variables

Variables store data that your program can use. **Global variables** are defined outside of functions and can be accessed anywhere:

```
a = 1
b = 2
print(a)  # 1
print(b)  # 2
```

Notes:

- Global variables can be accessed and modified from any part of the program.
- Avoid excessive use of globals; prefer **local variables** inside functions to prevent conflicts.

---

5. Data Types

Suny is **dynamically typed**, which means variables do not need an explicit type declaration. Suny automatically determines the type based on the value.

5.1 Boolean

Booleans represent truth values:

```
is_sunny = true
is_raining = false
print(is_sunny)   # true
print(is_raining) # false
```

Use in conditions:

```
weather = "sunny"
if weather == "sunny" do
    print("Go outside!")
else
    print("Stay inside!")
end
```

5.2 Numbers

Suny supports **integers** and **floating-point numbers**:

```
# Integers
a = 10
b = -5

# Floating-point numbers
c = 3.14
d = -0.5
```

Arithmetic Examples:

```
x = 10
y = 3
print(x + y)  # 13
print(x - y)  # 7
print(x * y)  # 30
print(x / y)  # 3.3333
```

5.3 Strings

Strings are sequences of characters enclosed in double quotes `" "`:

```
name = "Dinh Son Hai"
greeting = "Hello, world!"
print(name)     # Dinh Son Hai
print(greeting) # Hello, world!
```

Operations with Strings:

```
first = "Hello"
second = "World"
combined = first + " " + second
print(combined) # Hello World

text = "Suny"
print(size(text)) # 4
```

Using strings in conditions:

```
password = "1234"
if password == "1234" do
    print("Access granted")
else
    print("Access denied")
end
```

Escape Characters:

Escape	Meaning	Example	Output
`\n`	Newline	`"Hello\nWorld"`	Hello World
`\t`	Tab	`"Col1\tCol2"`	Col1 Col2
`\\`	Backslash	`"C:\\Path\\File"`	C:\Path\File
`\"`	Double quote	`"He said: \"Hi\""`	He said: "Hi"
`\'`	Single quote	`'It\'s sunny'`	It's sunny

5.4 Lists

Lists store multiple items:

```
numbers = [1, 2, 3, 4, 5]
names = ["Alice", "Bob", "Charlie"]
mixed = [1, "Two", true, 4.5]
```

Accessing items:

```
print(numbers[0])  # 1
print(names[2])    # Charlie
```

Modifying items:

```
numbers[0] = 10
print(numbers[0])  # 10
```

Adding and removing items:

```
push(numbers, 6)
pop(numbers)
```

Length of a list:

```
print(size(numbers))
```

Looping over lists:

```
fruits = ["apple", "banana", "cherry"]

# Using index
for i in range(size(fruits)) do
    print(fruits[i])
end

# Using item directly
for fruit in fruits do
    print(fruit)
end
```

5.5 Functions

Functions in Suny are **first-class values**: they can be assigned to variables, passed as arguments to other functions, returned from functions, and even created anonymously.

Basic Function

```
function add(a, b) do
    return a + b
end

print(add(1, 2))  # 3
```

Explanation:

- `function name(parameters) do ... end` defines a named function.

- `return` specifies the value that the function gives back.
- You can call the function using its name, passing required arguments.

Higher-Order Functions

Functions can accept other functions as arguments, or return functions:

```
function apply(func, x, y) do
    return func(x, y)
end

print(apply(add, 5, 7))  # 12
```

- `apply` takes a function `func` and two numbers, and calls `func(x, y)`.
- This demonstrates **higher-order functions**, useful for functional programming patterns.

Inner Functions

Functions can be **defined inside another function**. The inner function is local to the outer function:

```
function foo() do
    function bar() do
        print("This is bar")
    end

    return bar()
end

foo()  # Output: This is bar
```

- `bar` exists only within `foo`.
- This allows **encapsulation** and avoiding global namespace pollution.

Anonymous Functions

Anonymous functions are **functions without a name**. They can be used inline or stored in variables:

```
a = 10

print(function() do
```

```
    return a
end)()  # Output: 10
```

Explanation:

- `function() do ... end` defines an anonymous function.
- The function can be called immediately by adding `()` at the end.
- You can also assign it to a variable:

```
getA = function() do
    return a
end

print(getA())  # 10
```

Use Cases:

- Passing functions to other functions (callbacks).
- Returning functions from functions.
- Writing concise code without naming every function.

---

6. Logical Expressions

Logical operators:

- `and` → true if both values are true
- `or` → true if at least one value is true
- `not` → inverts the Boolean value

```
x = true
y = false
print(x and y) # false
print(x or y)  # true
print(not x)   # false
```

Using logical expressions in conditions:

```
is_sunny = true
has_umbrella = false
```

```
if is_sunny or has_umbrella do
    print("Go outside")
else
    print("Stay inside")
end

if not is_sunny do
    print("It is cloudy")
end
```

---

7. Control Structures

Conditional Statements

```
score = 75
if score >= 50 do
    print("You passed!")
else
    print("Try again")
end
```

While Loops

```
count = 1
while count <= 5 do
    print(count)
    count = count + 1
end
```

For Loops

Range-based:

```
for i in range(0, 5) do
    print(i)
end
```

Collection-based:

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits do
    print(fruit)
end
```

---

8. Summary

Suny provides a **clear and easy-to-learn syntax**, suitable for beginners and lightweight scripting tasks. With support for:

- Variables and data types
- Functions and higher-order functions
- Lists and loops
- Logical expressions and conditional statements

You can start building simple to

moderately complex programs **quickly and efficiently**.

The End

Thank you for reading!
by **dinhsonhai132**