# TinyPy Language Syntax Guide

For Developers

June 25, 2025

**Abstract**

This document serves as a comprehensive syntax guide for the TinyPy programming language, designed for developers working with the TinyPy-to-Python transpiler. It covers all language constructs, including data types, functions, control flow, input/output, data structures, operators, and comments, with examples and notes on behavior.

## 1 Introduction

TinyPy is a statically and dynamically typed programming language with C++-style syntax, designed to be transpiled into Python. This guide details its syntax, providing examples and explanations for developers. The language supports functions, variables, control flow, arrays, dictionaries, and input/output operations, with Python-style identifiers and automatic entry-point detection for `main()` functions.

## 2 Data Types

### 2.1 Static Typing

`int, float, bool, char, string`

**Description**: TinyPy supports five static data types: `int` (integers), `float` (floating-point numbers), `bool` (true/false), `char` (single characters), and `string` (text). These types enforce strict typing for variables, arrays, and function parameters/returns.

**Example**:

```
int x = 5;
float y = 3.14;
bool flag = true;
char c = 'A';
string s = "Hello";
```

### 2.2 Dynamic Typing

`dyn`

**Description**: The `dyn` keyword denotes dynamic typing, allowing variables or arrays to hold values of any type, similar to Python's dynamic typing.

**Example**:

```
dyn var = 42; // Can later be reassigned to "text" or true
```

## 3 Variables and Constants

### 3.1 Variable Declarations

`type var_name [= expression];`

**Description**: Variables are declared with a type (`int`, `float`, `bool`, `char`, `string`, `dyn`) and an optional initial value. Uninitialized variables are set to `None` in Python.

**Example**:

```
int x = 10;
dyn y; // Transpiles to y = None
```

### 3.2 Constants

`brick var_name = expression;`

**Description**: Constants are declared with the `brick` keyword and must be initialized. They behave like regular Python assignments but imply immutability.

**Example**:

```
brick MAX = 100;
```

### 3.3 Global Variables

`universal var_name;`

**Description**: The `universal` keyword declares global variables, transpiled to Python's `global` keyword.

**Example**:

```
universal count;
```

## 4 Functions

### 4.1 Function Definitions

`type func_name(param1, param2, ...) {`

**Description**: Functions are defined with a return type (`int`, `float`, `bool`, `char`, `string`, `dyn`) and optional parameters (typed or untyped). The body is enclosed in braces. The `main()` function triggers automatic execution in Python via `if` $_{name_{=="}{}_{main",:}}$.

**Example**:

```
int add(int a, int b) {
    ret a + b;
}
```

## 4.2 Return Statements

`ret expression;`

**Description**: The ret keyword returns a value from a function, transpiled to Python's return.

**Example**:

```
ret 42;
```

## 5 Input/Output

### 5.1 Printing

`disp << expression [, expression ...];`

**Description**: Outputs expressions (strings, variables, or numbers) to the console, separated by commas. Semicolons are optional and stripped during transpilation.

**Example**:

```
disp << "Hello" << x << 42;
```

### 5.2 User Input

`enter("%i|%f|%b|%c|%s|%dy", var | arr[index] | dict[key]);`

**Description**: Reads input into a variable, array element, or dictionary key using format specifiers: %i (int), %f (float), %b (bool), %c (char), %s (string), %dy (dynamic).

**Example**:

```
int x;
enter("%i", x);
enter("%i", numbers[0]);
enter("%s", student["name"]);
```

## 6 Control Flow

### 6.1 If Statements

```
thereBe {
    // statements
}if(condition)
```

**Description**: Initiates an if block with statements followed by a condition. The condition uses logical/comparison operators.

**Example**:

```
thereBe {
    disp << "Positive";
}if(x > 0)
```

### 6.2 Else If Statements

```
thereBe {
    // statements
}else if(condition)
```

**Description**: Continues a conditional block with an else-if clause.

**Example**:

```
thereBe {
    disp << "Zero";
}else if(x == 0)
```

### 6.3 Else Statements

```
alas {
    // statements
}
```

**Description**: Executes statements if no preceding conditions are true.

**Example**:

```
alas {
    disp << "Negative";
}
```

## 7 Loops

### 7.1 For Loop

```
repeatFor(type var_name = start; var_name < | <= | > | >= | != limit; var_name++) {
```

**Description**: A C-style for loop with initialization, condition, and increment (only ++ supported).

**Example**:

```
repeatFor(int i = 0; i < 10; i++) {
    disp << i;
}
```

### 7.2 While Loop

`repeatWhile(condition) {`

**Description**: Executes statements while the condition is true.

**Example**:

```
repeatWhile(x > 0) {
    x--;
}
```

### 7.3 Foreach Loop

`for(var_name : collection) {`

**Description**: Iterates over elements in an array or list, Python-style.

**Example**:

```
string names[2] = {"Alice", "Bob"};
for(name : names) {
    disp << name;
}
```

### 7.4 ForDict Loop

`forDict(key_var, value_var : dict) {`

**Description**: Iterates over key-value pairs in a dictionary.

**Example**:

```
dict students <int,string>[3] = {1: "Alice", 2: "Bob", 3: "Charlie"
   };
forDict(id, name : students) {
    disp << id << name;
}
```

## 8 Data Structures

### 8.1 Arrays

`type var_name[size] [= {value1, value2, ...}];`

**Description**: Arrays are declared with a type (int, float, bool, char, string, dyn) and a size (constant or variable). For dyn, arrays can hold mixed types, like Python lists. Initialization is optional.

**Example**:

```
int a[10];
int b[3] = {1, 2, 3};
dyn c[2] = {1, "S"};
```

## 8.2 Dictionaries

`dict var_name <key_type, value_type>[size] [= {key1: value1, key2: value2, ...}];`

**Description**: Dictionaries are dynamically typed, mutable, and declared with key and value types. Size is predefined, but initialization is optional.

**Example**:

```
dict student <string,int>[2];
dict students <int,string>[3] = {1: "Alice", 2: "Bob", 3: "Charlie"
    };
```

# 9 Operators

## 9.1 Increment/Decrement

`var_name++; var_name--;`

**Description**: Increments or decrements a variable by 1.

**Example**:

```
x++;
x--;
```

## 9.2 Conditional Operators

`&& (and), || (or), ! (not), == (equals), != (not equals), >, <, >=, <=`

**Description**: Used in conditions for logical and comparison operations. See precedence table below.

**Example**:

```
if(x > 0 && y != 0) {
    disp << "Valid";
}
```

### 9.3 Operator Precedence

| Operator | Description |
|---|---|
| `()` | Parentheses (highest precedence) |
| `!` | Logical NOT (right-to-left) |
| `==, !=, >, <, >=, <=` | Comparison (left-to-right) |
| `&&` | Logical AND (left-to-right) |
| `\|\|` | Logical OR (lowest precedence) |

## 10 Comments

`// comment`

**Description**: Single-line comments start with // and are ignored during transpilation

**Example**:

```
// This is a comment
```

**Note**: Multi-line comments are not supported in TinyPy.

## 11 Miscellaneous

### 11.1 Semicolon Endings

`statement;`

**Description**: Statements end with semicolons, which are stripped during transpilation to Python.

**Example**:

```
x = 5;
disp << x;
```

### 11.2 Python-Style Identifiers

`identifier`

**Description**: Identifiers follow Python's naming rules (letters, digits, underscores; cannot start with a digit).

**Example**:

```
int my_variable = 42;
```

### 11.3 Entry-Point Detection

`int main() {`

**Description**: If a main() function is defined, the transpiler adds if $_name_{=="}{}_main_{,,:main()}{}^{tot}$

**Example**:

```
int main() {
    disp << "Hello, TinyPy!";
    ret 0;
}
```