ZEN

Language Reference Manual

Eleanor Murguia (egm2142)
Zoe Gordin (zeg2103)
Nadia Saleh (ns3059)

Introduction

ZEN is language that allows users to write algorithms to create fractals and other geometric patterns. ZEN implements most standard C types and operators. ZEN follows a Java-like syntax but refrains from being object-oriented. Instead, ZEN provides built-in functions that allows users to create and combine geometric shapes.

Keywords

Keyword	**Description**
`while`	While loop
`for`	For loop
`if`	If statement
`else`	Else statement
`return`	Return function expression
`func`	Declare a function

Comments

Comments are single-line, and are indicated with the “#” symbol.

Delimiters

delimiter	**usage**
`()`	encloses tuples, defines order of operation, and contains arguments of function calls
`[]`	array initialization, assignment, and access
`{}`	scopes code
`;`	end of statement
`,`	separates elements in tuples and arrays, and arguments in function calls
`whitespace`	for readability only, ignored by compiler

Operators

In order of decreasing precedence

operator	**associativity**	**description**
`() []`	left-to-right	function call, list indexing
`! -`	right-to-left	logical and numerical negation
`* / %`	left-to-right	multiplication, division, modulo
`+ -`	left-to-right	addition, subtraction
`!= == <= >= < >`	left-to-right	boolean not equal, equal, less than or equal, greater than or equal, less than, greater than

`and or`	left-to-right	logical AND and OR
`=`	right-to-left	assignment
`.`	left-to-right	accessing methods of built in types

Primitive Types

type	**description**	**example**
`int`	integer	`int x = 2`
`float`	float, must include a digit before and after the decimal place	`float x = 2.5`
`bool`	single byte boolean	`false`
`string`	string	`“zen”`

Non-Primitive Types

`tuple`	a pair of elements	`(x, y)`
`list`	left-to-right	`[2, 4, 8]`

Operations for Tuples

Tuples provide two functions to get the x and y values, respectively:

```
tup.getX()
tup.getY()
```

Operations on Lists

Lists provide operations to return the length, modify the list, and return values from the list:

```
li.length()
li.get(int idx)
li.remove(int idx)
li.add(type element)
```

Control Flow

Statements

Statements include variable declaration and assignment, and always end with a semicolon (“;”).

```
float flo = 3.14;
```

Conditionals and Loops

If/else statements, for loops, and while loops are all standard. If statements do not require an `else` following them.

```
if(x != 5)
{
     y = y + 1;
}
else
{
     y = y + 2
}
```

Conditionals and loops are always enclosed in curly braces (“{}”). Both `for` loops and `while` loops are included in the language.

```
int i;
float flo = 1.5;
for(i = 0; i<3; i++)
```

```
{
     flo = flo + 0.5;
}


int a = 5;
while(a < 7)
{
     a = a + 2;
}
```

Functions

User Defined Functions

Users of ZEN can define their own functions with the following syntax:

```
func function_name(type1 arg1, type2 arg2){ #function body
}
```

Functions do not have a return type explicitly declared, but users can use the `return` keyword to return a value from the function.

Built In Functions

In addition to user defined function, ZEN has several built in function to assist in the creation of drawing shapes and fractals.

```
make_circle(int radius)
```

`make_circle` takes an integer corresponding to the radius of the circle that will be drawn.

```
make_ngon(int sides, int height, int width)
```

`make_ngon` takes three integers corresponding the number of sides of the ngon and the height and width of the ngon to be drawn.

ZEN also includes utility functions:

```
print(string output)
```

`print` takes one string that will be printed out to the console.