User Manual

Introduction

Cobra is an Open Source Graphical Output Language that will allow end users to develop algorithmic reasoning through visual programming. Cobra uses a simple syntax to be able to read code easily, but also write it thanks to the absence of delimiters such as semicolons. Cobra has a simple interface to help the user open their cobra programs and run them.

Installation

Binaries

For windows it is recommended use the “setup.exe”. It will install Project Cobra in the route **“C:\Project Cobra”**, and will create a start icon.

Source code

You can also download the source and run in windows, macOS and other operating systems.
* If you download the Source, you must have previously installed **python 2.7.x** and **pip**

For python
https://www.python.org/downloads/

For pip
https://pip.pypa.io/en/stable/installing/

Once you have **python 2.7** installed and **pip** you have to enter the following command in terminal or cmd:

```
pip install ply
```

Then you can download the source code of Project Cobra
Source Code
https://github.com/davilajose23/ProjectCobra

Usage

Programs can be written using any text editor or using the Integrated Development Environment that comes with the installation.

Open the terminal and move where Project Cobra was downloaded

Compilation with terminal

In order to compile programs you have to run the following command:

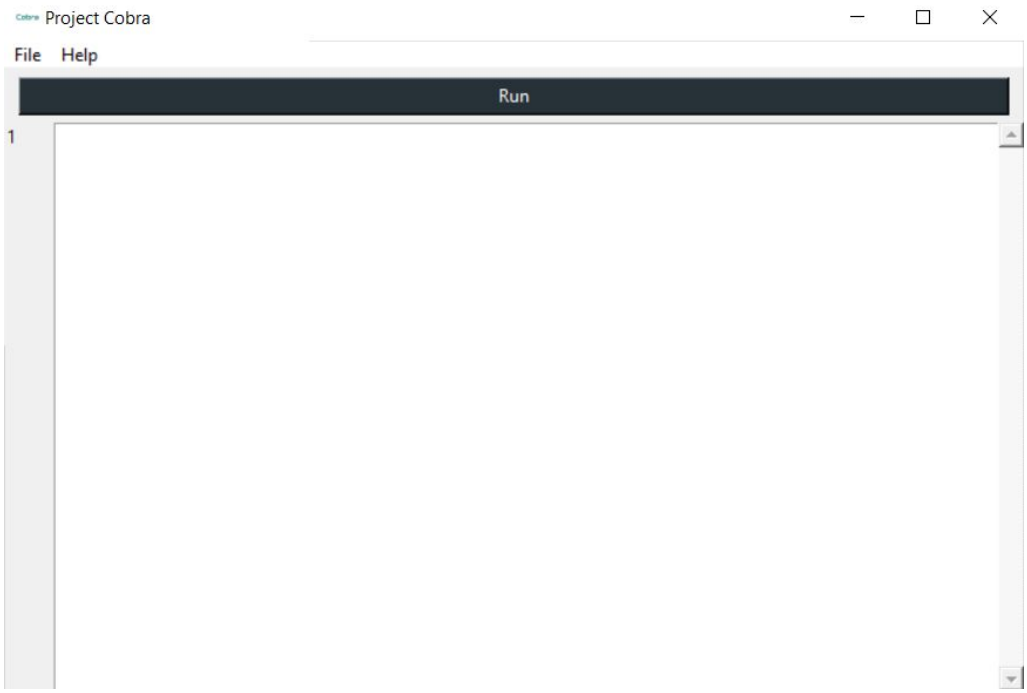
```
python parser.py program.cbr
```

IDE

To use the IDE you have to run the following command:

```
python ide.py
```

you will see a window like this:



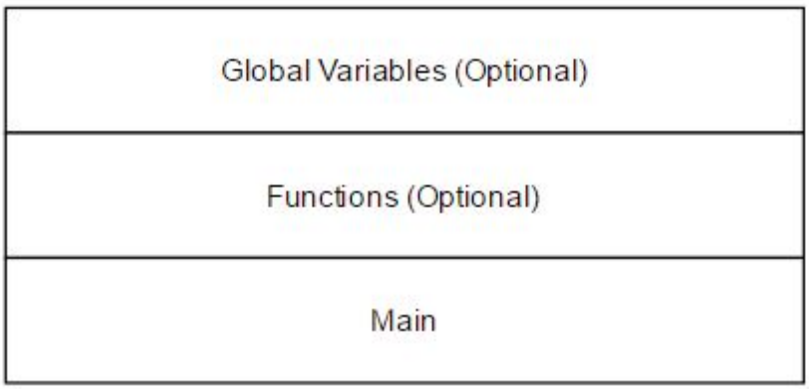
You can write your code in the editor area and when you are ready click Run button.
***Make sure you don't close terminal window**

Tutorial

This tutorial is designed for all people that want to learn Project Cobra.
*You must have Project Cobra Installed, see the Installation section for more details

Basic Struct

All programs will have the following structure.



Let's notice that only the Main is not optional

```
# ------------------Global variables Section ------------------------
int a

# -------------------Functions Section -----------------------------
func int suma(int a, int b):
      return a + b
end

#---------------------- Main Section ----------------------------------
main
       int b
       a = 4
       b = 5
       print suma(a, b)
endmain
```

```
>>> 9
```

First Program

Let's write our first hello world program:
(Hello_World.cbr)

```
main
        print "Hello World"
endmain
```

```
>>> Hello World
```

Reserved Words

The following list show the Project Cobra keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Project Cobra keywords contain lowercase letters only.

- **if**
- **end**
- **while**
- **print**
- **read**
- **else**
- **and**
- **or**
- **for**
- **return**
- **true**
- **false**
- **bool**
- **int**
- **double**
- **string**
- **void**
- **from**
- **to**
- **mod**
- **main**
- **endmain**
- **func**

Comments

The comments in Project Cobra start with a **#** and end with and EOL.

```
# my first program
main
        # this is a comment
        print "Hello World"
endmain
```

Types

There are 4 basic types:

Int

Integer of 32 or 64 bits depending on the system.

Double

Maximum value 1.7976931348623157e+308

String

Any sequence of characters contained between ‘’ or “”

Bool

true or false

Variables

There are only 3 sections to declare variables.

Global

Only at the beginning of the code you can declare global variables, before the functions definition.

```
# ------------------Global variables Section ------------------------
int a
int cont
bool flag
# -------------------Functions Section -------------------------------
func ...
    ...
end
#---------------------- Main Section ---------------------------------
main
   ...
endmain
```

In Functions

Inside a function you can only declare variables at the beginning, before writing any code.

```
# ------------------Global variables Section -------------------------
...
# -------------------Functions Section -------------------------------
func int suma(int a):
    int variable
    bool flag

    ...
end
#---------------------- Main Section --------------------------------
main
   ...
endmain
```

In the Main

Inside the Main you can only define variables at the beginning before writing any code.

```
# ------------------Global variables Section -------------------------
...
#---------------------- Main Section --------------------------------
func ...
    ...

end
#---------------------- Main Section --------------------------------
main
    int variable
    bool flag
   ...
endmain
```

Vectors

In Project Cobra there is a data structure named vector. This structure holds a collection of values that can be accessed by the index.

Declaration

The basic structure for declaration is the following:

```
TYPE name [ SIZE ]
```

Example

```
int a[5]
bool cont[10]
```

Access

In your code, you can access the value of a vector by their index. The basic structure is the following:

```
name [ INDEX ]
```

* you can use a vector like any other variable for expressions

Example

```
b = a[5] * 10
print cont[10] + b
```

I/O

The basic I/O functions that Project Cobra are read() and print

Read(output)

Reads the user input one value at at time, meaning that it will only be able to store a value to a variable and must be repeated to read a sequence of values. The user must be aware of the type of input they are giving to the function and the type of the variable that intends to store the given value.

```
int a
read(a)
```

Print (Input)

Prints to the terminal the received value (meaning integer or double constants in the form of 1, or 1.0 respectively), the value of a variable or a string constant.

- 1st argument is the value to display

```
main
	print “Hello World”
	print “Hello”
endmain
```

```
>>> Hello World
>>> Hello
```

- 2nd argument of type string is the end of the print (this argument is optional and will be displayed at the end)

```
main
	print “Hello World ”, “- ”
	print “Hello”
endmain
```

```
>>> Hello World - Hello
```

The following program will wait for user Input and display it.

```
main
        int age
        print “please enter your age”
        read(age)
        print “your age is ”, “”
        print(age)
endmain
```

```
>>> please enter your age
>>> 9
>>> your age is 9
```

Operators

All the operators have left association meaning that they should have an operand to their left in order to be valid.

Arithmetic Operators

Let’s assume that **“a”** variable has a value of 20 and “**b**” variable has a value of 5

Operator	Description	Example (Code)	Output
+	Add values	`print a + b`	`>>> 25`
-	Substract value b form a	`print a - b`	`>>> 15`
*****	Multiply values	`print a * b`	`>>> 100`
/	Divide a by b	`print a / b`	`>>> 4`
%	Obtain the module of a by b	`print a % b`	`>>> 0`
mod	Obtain the module of a by b	`print a mod b`	`>>> 0`

* Note that there is important to have a space after the operand (it may cause problems with operan **“-”** if there is not a space)

Relational Operators

Let's assume that **"a"** variable has a value of 20 and "**b"** variable has a value of 5

Operator	Description	Example (Code)	Output
==	Check if a has the same value as b	`a == b`	`>>> False`
!=	Check if a and b have different values	`a != b`	`>>> True`
<	Check if a is less than b	`a < b`	`>>> False`
>	Check if a is greater than b	`a > b`	`>>> True`
>=	Check if a is greater or equal to b	`a >= b`	`>>> True`
<=	Check if a is less or equal to b	`a <= b`	`>>> False`
and	Check if a is less than b AND greater than 5	`a < b and a > 5`	`>>> False`
or	Check if a is less than b OR if a is greater than 5	`a < b or a > 5`	`>>> True`

Assignment Operators

Let's assume that **"a"** variable has a value of 20 and "**b"** variable has a value of 5

Operator	Description	Example (Code)	Equivalecy
=	Assigns the value 20 to variable a	`a = 20`	-
+=	Assigns the value of a + b to a	`a += b`	`a = a + b`
-=	Assigns the value of a - b to a	`a -= b`	`a = a - b`

***=**	Assigns the value of a * b to a	`a *= b`	`a= a * b`
/=	Assigns the value of a / b to a	`a /= b`	`a = a / b`

Conditions

The **if** keyword is used to execute a block of code, if, and only if a condition is true.
The syntax is

```
if (condition):
   ...
end
```

You can add an **else** statement that will execute when the **if** condition is false:

```
if (condition):
    ...
else:
     ...
end
```

Ejemplo

```
int a
int b
main
     a = 20
     b = 5
     if (a > 5):
          print “if statement”
     else:
          print “else statement”
     end
endmain
```

```
>>> if statement
```

Functions

Declaration

In project cobra you can declare your own functions. The basic structure is the following:

```
func TYPE name( TYPE param_name, TYPE param_name):
     ...
     return value
end
```

* The functions can or can't have parameters
The return statement is optional depending on the TYPE of the function

Calls

The functions you defined can be called by the following structure:

```
name(argument1, argument2)
```

* The functions can or can't have arguments

Example

```
func int suma(int a, int b):
     return a + b
end

main
    print suma(5, 9)
endmain
```

```
>>> 14
```

Cycles

There are two types of cycles: For and while

for

This is the structure of the for

```
for variable_name from inferior_limit  to superior_limit :
     ...
end
```

Example

```
for i from 0 to 5:
   print i
end
```

```
>>> 0
>>> 1
>>> 2
>>> 3
>>> 4
```

* Notice that the inferior limit is inclusive but superior limit is not.
* Also **inferior_limit < superior_limit**

while

This is the structure of the while

```
while condition :
   ...
end
```

Example

```
int a
a = 5
while  a > 0:
       print a
       a -= 1
end
```

```
>>> 5
>>> 4
>>> 3
>>> 2
>>> 1
```

Draw Functions

`drawText(double x, double y, string text, int size, string color)`

Allows the user to print text in the graphical output window. It receives **5** arguments in the following order:

- **double x**: coordinate x of the center.
- **double y**: coordinate y of the center.
- **string text**: the expected string to represent in the graphical output.
- **int** size: determines the size of the text in the screen
- **string** color:

`drawLine(double ax, double ay, double bx, double by, int size, string fill)`

Allows users to print lines in the graphical output window. It receives **6** arguments in the following order:

- **double ax**: The x coordinate of the starting point of the line
- **double ay**: the y coordinate of the starting point of the line
- **double bx**: the x coordinate of the final point of the line
- **double by**: the y coordinate of the final point of the line
- **int size**: the size that determines the width.
- **string fill**: the color of the line

`drawCircle(double x, double y, double radio, int size, string fill, string line)`

Draws circles in the graphical output window. It receives **6** arguments in the following order:

- **double x:** is the x coordinate of the center of the circle.

- **double y:** is the y coordinate of the center of the circle.
- **double radio:** is the radio of the circle.
- **int size:** is the width of the line.
- **string fill:** is the color that goes inside the circle.
- **string line:** is the color of the line of the circle.

`drawOval(double ax, double ay, double bx, double ay, int size, string fill, string line)`

Similar to the drawCircle function, however it allows drawing non-uniform circles. It receives **7** arguments in the following order:

* The 2 pairs are similar to receiving a rectangle points. It's expected an upper-left corner coordinate and lower-right coordinate. This determines the width and height of the oval.

- **double ax**: is the x coordinate of the upper-left corner.
- **double ay:** is the y coordinate of the upper-left corner.
- **double bx**: is the x coordinate of the low-right corner.
- **double by**: is the y coordinate of the low-right corner.
- **int size**: is the width of the line.
- **string fill**: is the color that goes inside the oval.
- **string line**: is the color of the line of the oval.

`drawTriangle(double ax, double ay, double bx, double by, double cx, double cy, int size, string fill, string line)`

Allows user to draw triangles in the graphical output window. It receives 9 arguments in the following order:

- double ax: is the x coordinate of the first point of the triangle
- double ay: is the y coordinate of the first point of the triangle
- double bx: is the x coordinate of the second point of the triangle
- double by: is the y coordinate of the second point of the triangle
- double cx: is the x coordinate of the third point of the triangle
- double cy: is the x coordinate of the third point of the triangle
- int size: is the size of the line
- string fill: is the color that goes inside the triangle
- string line: is the color of the line

`drawRectangle(double ax, double ay, double bx, double by, int size, string fill, string line)`

Allows the user to draw rectangles in the graphical output window. It receives 7 arguments in the following order:

- double ax: is the x coordinate of the up-left corner of the rectangle
- double ay: is the y coordinate of the up-left corner of the rectangle
- double bx: is the x coordinate of the down-right corner of the rectangle
- double by: is the y coordinate of the down-right corner of the rectangle
- int size: is the size of the line
- string fill: is the color that goes inside the rectangle
- string line: is the color of the line

`drawDot(double x, double y, string fill)`

Creates a dot in the graphical output window. It receives 3 arguments in the following order:

- double x: is the x coordinate of the dot.
- double y: is the y coordinate of the dot.
- string fill: is the color of the dot

`drawCurve(double ax, double ay, double bx, double by, fill)`

Draws a curve in the graphical output window. It receives 5 arguments in the following order:

- double ax: is the x coordinate of the starting point of the curve.
- double ay: is the y coordinate of the starting point of the curve.
- double bx: is the x coordinate of the finishing point of the curve.
- double by: is the y coordinate of the finishing point of the curve
- string fill: is the color of the curve.

`insertImage(double ax, double ay, string filename)`

Allows the user to insert an image in the graphical output window. It receives 3 arguments in the following order:

- double ax: is the x coordinate of the center of the image
- double ay: is the y coordinate of the center of the image
- string filename: the filename of the image with extension:

* Notice that the image must be placed in the same folder as the program. The compiler only accepts *.gif images.

Color

For most of the Draw Functions you have to specify a color as a string. Most normal color as “red”, “purple”, “blue”, “green”, “cyan”, etc. are available. There is a list of other name of some common colors http://cng.seas.rochester.edu/CNG/docs/x11color.html.

Also you can specify any color in hexadecimal format. For example: “#2196f3”
This is a tool to you to calculate more colors:
https://www.w3schools.com/colors/colors_hexadecimal.asp