

# COMPILER FOR BASIC DARTMOUTH 64

## MANUAL DE USUARIO

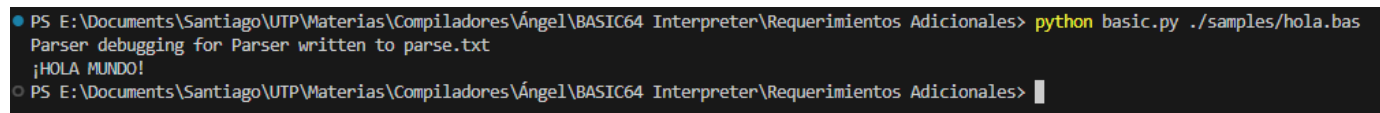
Para usar este compilador, se requiere obligatoriamente una instalación de Python en su ordenador, se recomienda utilizar la última versión si no está instalado, la cual puedes descargar desde la página oficial de Python. Una vez hecha la instalación, por la línea de comandos de tu sistema operativo o en el IDE de tu gusto que tenga acceso a terminal como Visual Code Studio, ejecuta el siguiente comando para instalar las librerías necesarias para el funcionamiento del compilador:

```
pip install -r requirements.txt
```

Después de haber instalado las librerías especificadas en ‘requirements.txt’, para ejecutar un programa de BASIC64 entre los que están en la carpeta ‘samples’, puedes escribir el siguiente comando a manera de ejemplo para ejecutar ‘hola.bas’, el cual se encuentra en ‘samples’. Puedes reemplazar ‘hola.bas’ por el programa de tu gusto y se sugiere abrir la línea de comandos/terminal o la IDE desde esta carpeta principal para evitar inconvenientes:

```
python basic.py ./samples/hola.bas
```

Este será el resultado de ejecutar dicho comando en tu terminal:



```
PS E:\Documents\Santiago\UTP\Materias\Compiladores\Ángel\BASIC64 Interpreter\Requerimientos Adicionales> python basic.py ./samples/hola.bas
Parser debugging for Parser written to parse.txt
¡HOLA MUNDO!
PS E:\Documents\Santiago\UTP\Materias\Compiladores\Ángel\BASIC64 Interpreter\Requerimientos Adicionales>
```

Dentro del archivo ‘basic.py’, se especifican varios argumentos con opciones de depuración como imprimir los tokens procesados por el analizador léxico o imprimir el árbol de sintaxis abstracto (AST) generado por el analizador sintáctico; también se especifican banderas que permiten modificar el funcionamiento del intérprete y como consecuencia, que el programa entregado se ejecute de forma diferente.

```

...
Usage: basic.py [-h] [-a style] [-o OUT] [-l] [-D] [-p] [-I] [--sym] [-S] [-R] [-u] [-ar] [-sl] [-n] [-g] [-t] [--tabs] input

Compiler for BASIC DARTMOUTH 64

Positional arguments:
  input          BASIC program file to compile

Optional arguments:
  -h, --help            Show this help message and exit
  -D, --debug           Generate assembly with extra information (for debugging purposes)
  -o OUT, --out OUT    File name to store generated executable
  -l, --lex            Store output of lexer
  -a STYLE             Generate AST graph as DOT or TXT format
  -I, --ir            Dump the generated Intermediate representation
  --sym              Dump the symbol table
  -S, --asm          Store the generated assembly file
  -R, --exec         Execute the generated program
  -v, --version      Show the version of the BASIC interpreter
  -u, --uppercase    Convert all entries to uppercase
  -ar INT, --array-base INT Set the minimum index of the dimensional arrays (default is 1)
  -sl, --slicing     Enable string slicing (disable string arrays)
  -n, --no-run       Don't run the program after parsing
  -g, --go-next      If no branch from a GOTO instruction exists, go to the next line
  -t, --trace        Activate tracing to print line numbers during execution
  --tabs INT         Set the number of spaces for comma-separated elements (default is 15)
  -rn INT, --random INT Set the seed for the random number generator
  -p, --print-stats  Print statistics on program termination
  -w, --write-stats  Write statistics to a file on program termination
  -of, --output-file Redirect PRINT output to a file
  -if INPUT_FILE, --input-file INPUT_FILE Redirect INPUT to a file
...

```

Para poder ejecutar un programa de BASIC usando una o más banderas (no se puede combinar la bandera de impresión de tokens y la bandera de impresión del árbol AST con el resto de banderas, únicamente), se escribe el comando de la siguiente manera. Por ejemplo, para ejecutar un programa de BASIC64 que requiera especificar el índice mínimo para los arreglos/dimensiones para que funcione correctamente:

```
python basic.py -ar 0 ./samples/game_of_life.bas
```

Después de escribir la bandera ‘-ar’, se indica inmediatamente el índice, el cual debe ser un valor numérico entero ‘INT’, en este caso el número ‘0’. De lo contrario, si no se especifica la bandera, pueden ocurrir errores durante la ejecución, como el siguiente al correr ‘game\_of\_life.bas’:

```
Indexes of A are out of bounds at line 325
```

Otros archivos complementarios para banderas como ‘-if’ que redirecciona la instrucción INPUT a un archivo, se encuentran dentro de la carpeta ‘samples’ y se invocan de la misma manera como se haría con un programa de BASIC.

```
python basic.py -if ./samples/game_input.txt ./samples/game.bas
```

```
PS E:\Documents\Santiago\UTP\Materias\Compiladores\Ángel\BASIC64 Interpreter\Requerimientos Adicionales> python basic.py -if ./samples/game_input.txt ./samples/game.bas
Parser debugging for Parser written to parse.txt
Redirecting INPUT to read from file: ./samples/game_input.txt
GUESS THE NUMBER BETWEEN 1 AND 100.
TOO SMALL, GUESS AGAIN
TOO SMALL, GUESS AGAIN
TOO LARGE, GUESS AGAIN
TOO LARGE, GUESS AGAIN
TOO SMALL, GUESS AGAIN
TOO SMALL, GUESS AGAIN
TOO SMALL, GUESS AGAIN
TOO SMALL, GUESS AGAIN
TOO LARGE, GUESS AGAIN
TOO LARGE, GUESS AGAIN
TOO SMALL, GUESS AGAIN
```

```
TOO SMALL, GUESS AGAIN
TOO SMALL, GUESS AGAIN
TOO LARGE, GUESS AGAIN
TOO LARGE, GUESS AGAIN
TOO SMALL, GUESS AGAIN
TOO SMALL, GUESS AGAIN
TOO SMALL, GUESS AGAIN
TOO SMALL, GUESS AGAIN
TOO LARGE, GUESS AGAIN
TOO LARGE, GUESS AGAIN
No more input data available in the file.
```

-----

A continuación, se describe cada archivo dentro de la carpeta. Si prefieres ver el resultado de alguna de las fases del compilador, puedes reemplazar ‘basic.py’ en el comando por cualquiera de estos archivos. Cabe aclarar que no se puede ejecutar ‘basinterp.py’ sin antes haber realizado el análisis léxico y sintáctico, ni tampoco puede ejecutarse ‘basrender.py’ sin antes haber realizado el análisis sintáctico, por lo que no se pueden ejecutar por sí solos. Para los demás, no se requiere el uso de banderas como en ‘basic.py’.

basic.py	- Compilador de BASIC64
bascontext.py	- Clase de alto nivel que contiene todo lo relacionado con el análisis y/o ejecución de un programa en BASIC64
baslex.py	- Analizador léxico de BASIC64
basparse.py	- Analizador sintáctico de BASIC64
basast.py	- Árbol AST de BASIC64
basrender.py	- Dibujante del árbol AST de BASIC64
basinterp.py	- Intérprete de BASIC64
ircode.py	- Generador de código intermedio
basinterpir.py	- Intérprete de código intermedio