ONAPSIS

¿CÓMO ES *HACER* UN LENGUAJE?

Parsing, compilación y ejecución de programas en Onapsis

Julio 2021 | Ignacio “mega” Losiggio

AGENDA

Motivación
¿Por qué hacemos análisis de programas en Onapsis?

Leyendo programas
Introducción al parsing usando ANTLR4

Entendiendo programas
Intérpretes sobre un AST.

Ejecutando programas
Máquinas virtuales (Stack machines y register machines)

MOTIVACIÓN

¿Por qué hacemos análisis de código en Onapsis?

MOTIVACIÓN: ¡Agreguemos algo a un lenguaje!

Vamos a comenzar desde un lenguaje con:

- Valores numéricos
- Valores booleanos
- Variables
- Condicionales (if/else)
- *Algunas* operaciones aritméticas

Nuestro objetivo: agregar **funciones** a un lenguaje de juguete

Vamos a recorrer el camino desde parsing hasta ejecución en una máquina virtual

LEYENDO PROGRAMAS

Introducción al parsing usando ANTLR4

LEYENDO PROGRAMAS: El lenguaje que ya tenemos

Miren estos ejemplos

```
escribir(10)
```

```
variable = 5
variable = variable + 5
escribir(variable)
```

```
valor = leer_num()
if valor >= 30 then
    escribir('Muy grande!')
else
    escribir('Ok!')
end
```

```
if leer_num() != 42 then
    escribir('Mal!')
end
```

¿Se les ocurren reglas que describan el lenguaje?

LEYENDO PROGRAMAS: Describiendo estas “reglas”

Intentemos formalizar lo que vimos:

- Condicionales (tienen dos formas):
  - if <condición> then <cuerpo> else <cuerpo> end
  - if <condición> then <cuerpo> end
- Llamadas a funciones
  - <nombre>(<argumento #1>, <argumento #2>, …)
- Asignaciones a variables
  - <nombre> = <valor>
- Expresiones aritméticas
  - <parte izquierda> + <parte derecha>

Más importante aún es lo que no vimos:

- ¿Es válido hacer algo cómo escribir(leer_num() + 3)?
- ¿Hay restas? ¿Multiplicación? ¿División?
- ¿Se pueden hacer nuevas funciones o sólo se pueden usar las que vienen “precargadas”?

LEYENDO PROGRAMAS: ANTLR

```
valor = leer_num()
if valor >= 30 then
    escribir('Muy grande!')
else
    escribir('Ok!')
end
```

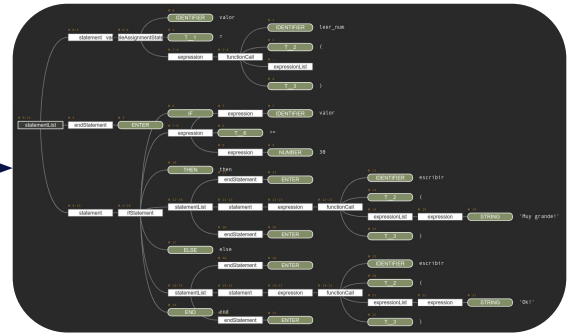
ejemplo.ona

```
class OnaParser(Parser):
    grammarFileName = …

    def __init__(self, …):
        …

    def if_statement(self):
        …
```

OnaParser.py



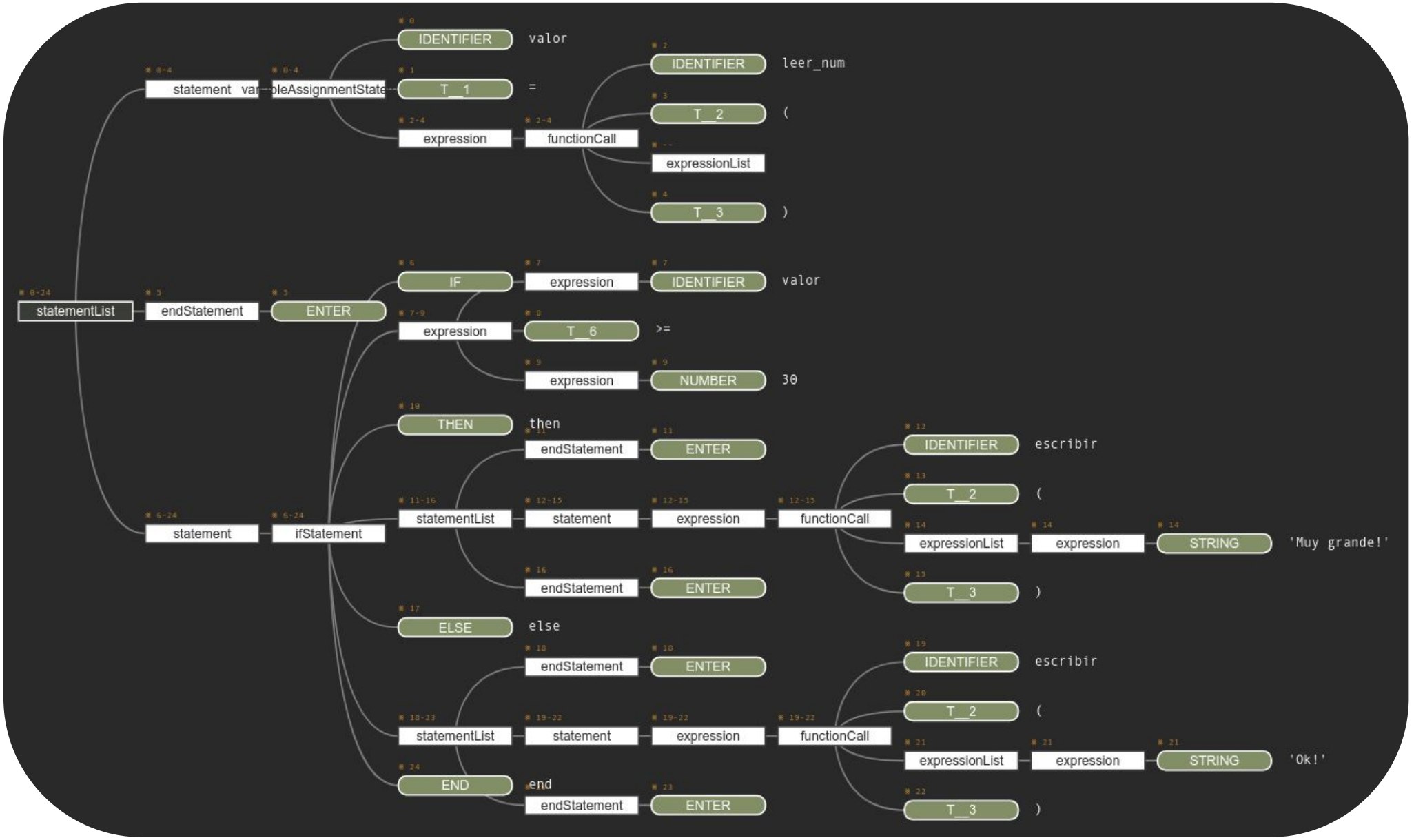
Parse Tree

```
IDENT
  : [A-Za-z_][A-Za-z0-9_]* ;
ifStatement
  : IF expression THEN statementList
    (ELSE statementList)?
    END
  ;
…
```

Ona.g4

ANTLR4

LEYENDO PROGRAMAS: ANTLR



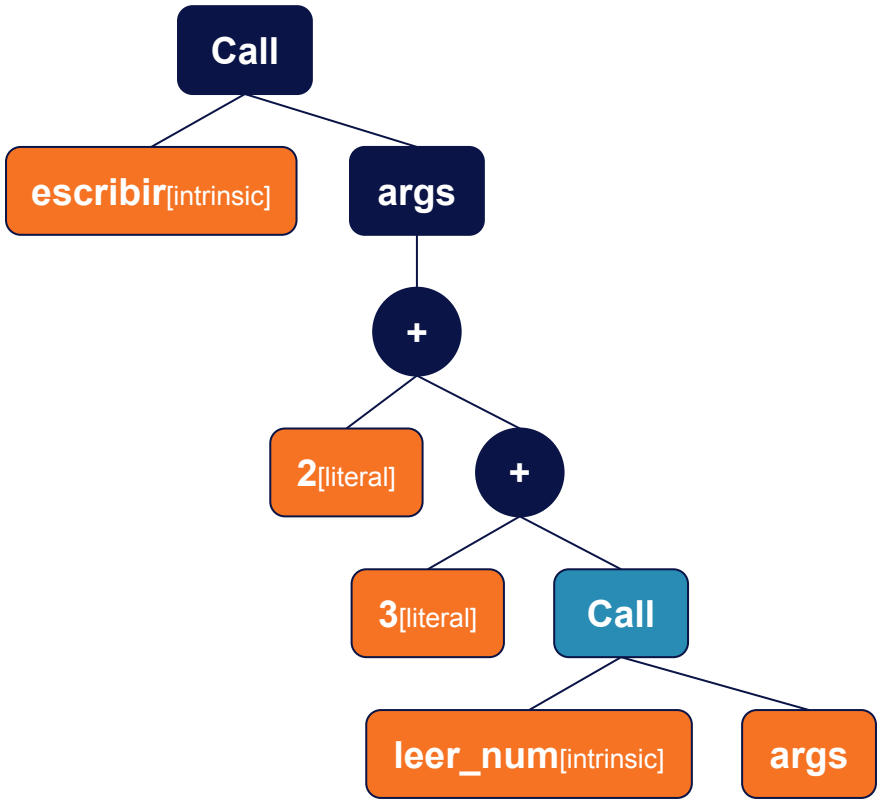
ENTENDIENDO PROGRAMAS

Intérpretes sobre un AST

ENTENDIENDO PROGRAMAS: Intérpretes sobre un AST

No es una técnica súper popular pero nos va a ayudar a entender el proceso de compilación.

```
escribir(2 + 3 + leer_num())
```

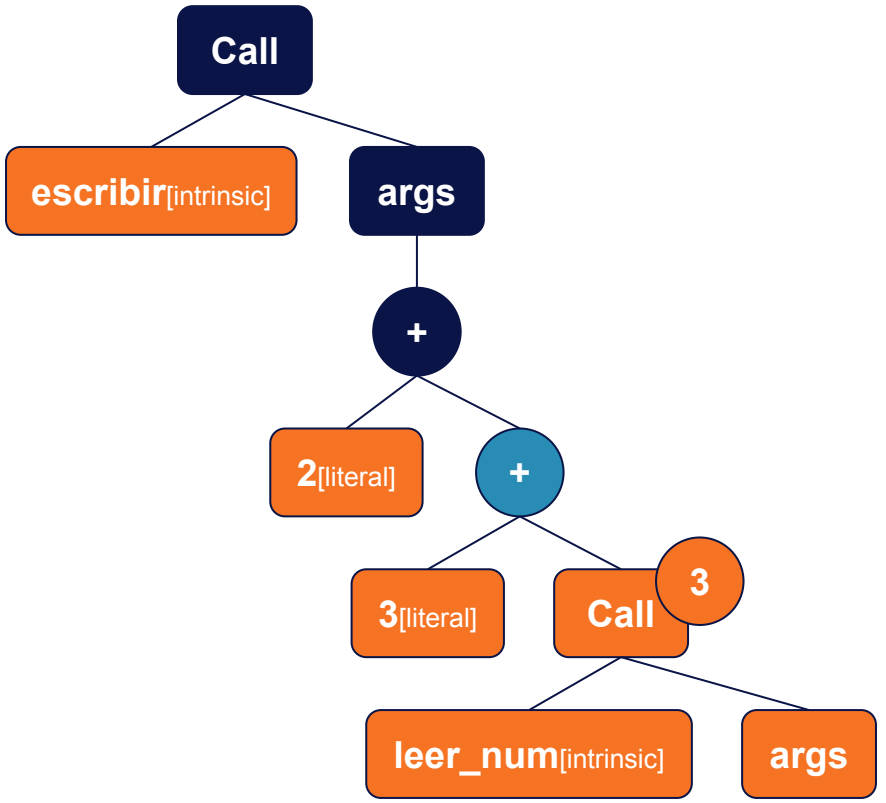



```
~/ECI2021$ ./interprete
ejemplo.ona
Output:
```

ENTENDIENDO PROGRAMAS: Intérpretes sobre un AST

No es una técnica súper popular pero nos va a ayudar a entender el proceso de compilación.

```
escribir(2 + 3 + leer_num())
```

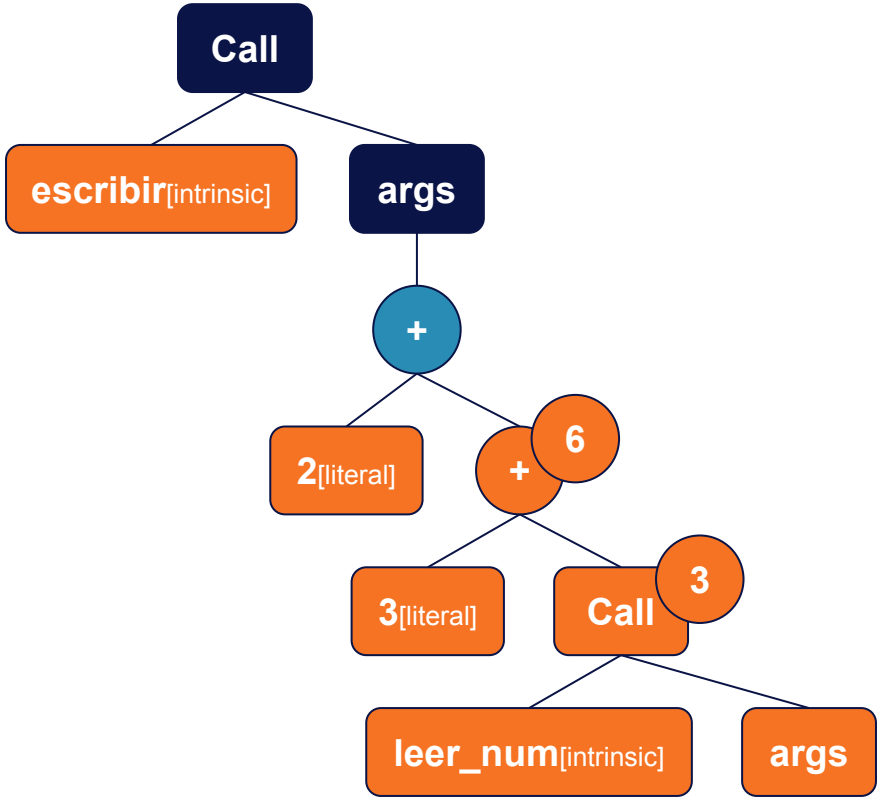



```
~/ECI2021$ ./interprete
ejemplo.ona
Output:
```

ENTENDIENDO PROGRAMAS: Intérpretes sobre un AST

No es una técnica súper popular pero nos va a ayudar a entender el proceso de compilación.

```
escribir(2 + 3 + leer_num())
```

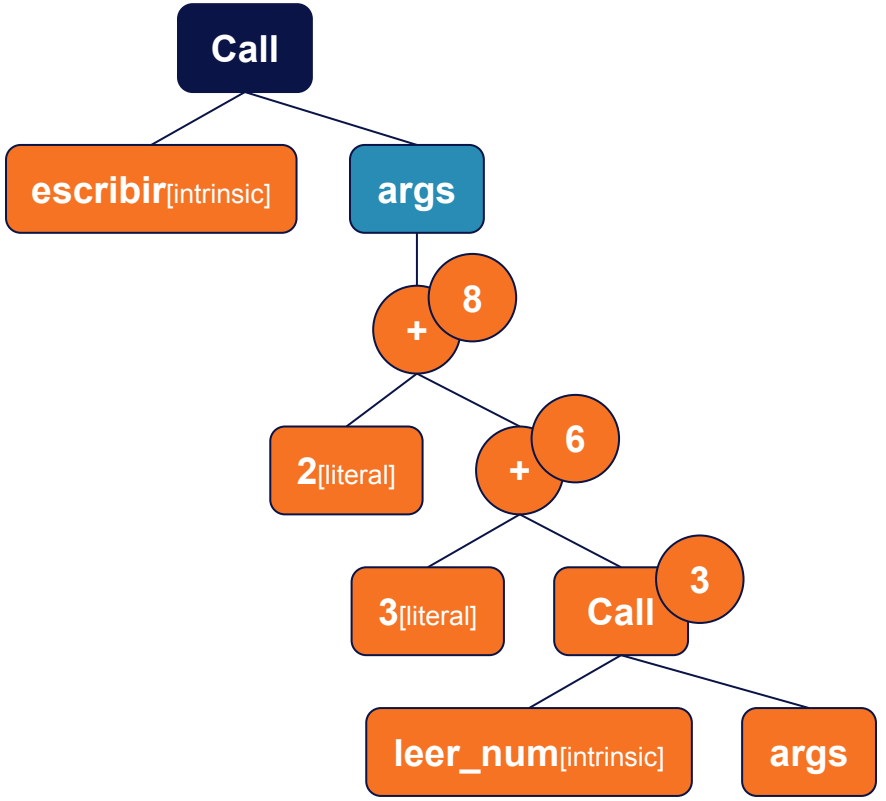



```
~/ECI2021$ ./interprete
ejemplo.ona
Output:
```

ENTENDIENDO PROGRAMAS: Intérpretes sobre un AST

No es una técnica súper popular pero nos va a ayudar a entender el proceso de compilación.

```
escribir(2 + 3 + leer_num())
```

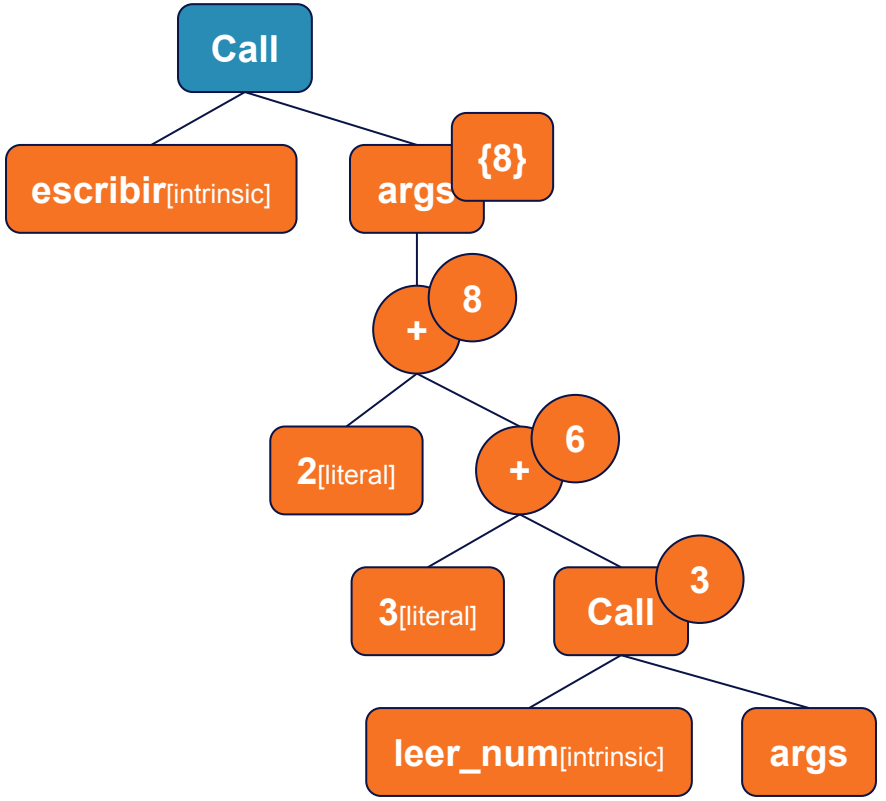



```
~/ECI2021$ ./interprete
ejemplo.ona
Output:
```

ENTENDIENDO PROGRAMAS: Intérpretes sobre un AST

No es una técnica súper popular pero nos va a ayudar a entender el proceso de compilación.

```
escribir(2 + 3 + leer_num())
```

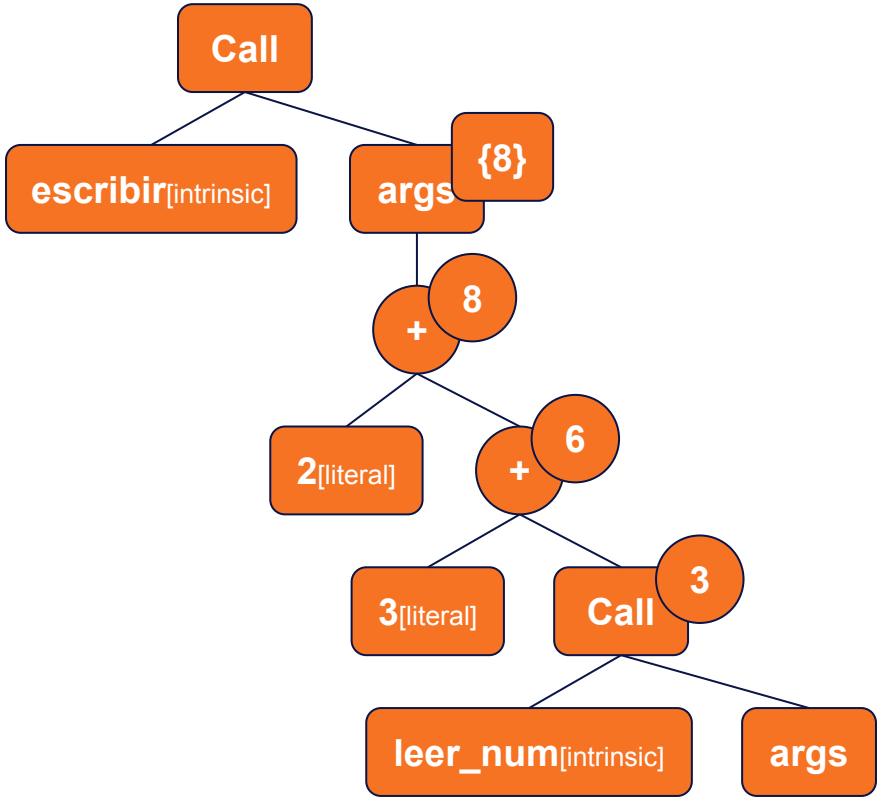



```
~/ECI2021$ ./interprete
ejemplo.ona
Output:
```

ENTENDIENDO PROGRAMAS: Intérpretes sobre un AST

No es una técnica súper popular pero nos va a ayudar a entender el proceso de compilación.

```
escribir(2 + 3 + leer_num())
```




```
~/ECI2021$ ./interprete
ejemplo.ona
Output:
8
```

ENTENDIENDO PROGRAMAS: Compilación

¿Podemos “aplastar” el árbol?

¿Cómo hacemos cuando la ejecución no es lineal?

(ifs, whiles, generadores, etc)

EJECUTANDO PROGRAMAS

Máquinas virtuales (stack machines y register machines)

```
/**
 * For the brave souls who get this far: You are the chosen ones,
 * the valiant knights of programming who toil away, without rest,
 * fixing our most awful code. To you, true saviors, kings of men,
 * I say this: never gonna give you up, never gonna let you down,
 * never gonna run around and desert you. Never gonna make you cry,
 * never gonna say goodbye. Never gonna tell a lie and hurt you.
 */
```

https://stackoverflow.com/a/482129 | *What is the best comment in source code you have ever encountered?*

EJECUTANDO PROGRAMAS: BYTECODES

```
function <pedir_num.lua:1,6> (19 instructions at 0x55e931e83960)
0 params, 5 slots, 1 upvalue, 0 locals, 8 constants, 0 functions
    1    [2]  GETTABUP 0 0 -1    ; _ENV "print"
    2    [2]  LOADK    1 -2      ; "Escrib\195\173 un n\195\186mero: "
    3    [2]  CALL     0 2 1
    4    [3]  GETTABUP 0 0 -4    ; _ENV "io"
    5    [3]  GETTABLE 0 0 -5    ; "read"
    6    [3]  CALL     0 1 2
    7    [3]  SETTABUP 0 -3 0    ; _ENV "num_texto"
    8    [4]  GETTABUP 0 0 -7    ; _ENV "tonumber"
    9    [4]  GETTABUP 1 0 -3    ; _ENV "num_texto"
    10   [4]  CALL     0 2 2
    11   [4]  SETTABUP 0 -6 0    ; _ENV "num"
    12   [5]  GETTABUP 0 0 -1    ; _ENV "print"
    13   [5]  GETTABUP 1 0 -6    ; _ENV "num"
    14   [5]  LOADK    2 -8      ; "al cuadrado es"
    15   [5]  GETTABUP 3 0 -6    ; _ENV "num"
    16   [5]  GETTABUP 4 0 -6    ; _ENV "num"
    17   [5]  MUL      3 3 4
    18   [5]  CALL     0 4 1
    19   [6]  RETURN   0 1
```

EJECUTANDO PROGRAMAS: BYTECODES

```
[generated bytecode for function: pedir_num]
Parameter count 1
Frame size 56
   18 E> 0x26801c4b0232 @    0 : a0             StackCheck
   43 S> 0x26801c4b0233 @    1 : 13 00 00       LdaGlobal [0], [0]
         0x26801c4b0236 @    4 : 26 f9          Star r2
         0x26801c4b0238 @    6 : 12 01          LdaConstant [1]
         0x26801c4b023a @    8 : 26 f8          Star r3
   43 E> 0x26801c4b023c @   10 : 5b f9 f8 02    CallUndefinedReceiver1 r2, r3, [2]
         0x26801c4b0240 @   14 : 26 fb          Star r0
   86 S> 0x26801c4b0242 @   16 : 13 02 04       LdaGlobal [2], [4]
         0x26801c4b0245 @   19 : 26 f9          Star r2
   86 E> 0x26801c4b0247 @   21 : 5b f9 fb 06    CallUndefinedReceiver1 r2, r0, [6]
         0x26801c4b024b @   25 : 26 fa          Star r1
  110 S> 0x26801c4b024d @   27 : 13 03 08       LdaGlobal [3], [8]
         0x26801c4b0250 @   30 : 26 f8          Star r3
  118 E> 0x26801c4b0252 @   32 : 28 f8 04 0a    LdaNamedProperty r3, [4], [10]
         0x26801c4b0256 @   36 : 26 f9          Star r2
         0x26801c4b0258 @   38 : 12 05          LdaConstant [5]
         0x26801c4b025a @   40 : 26 f6          Star r5
         0x26801c4b025c @   42 : 25 fa          Ldar r1
  149 E> 0x26801c4b025e @   44 : 34 fa 0c       Mul r1, [12]
         0x26801c4b0261 @   47 : 26 f5          Star r6
         0x26801c4b0263 @   49 : 27 fa f7       Mov r1, r4
  118 E> 0x26801c4b0266 @   52 : 55 f9 f8 04 0d CallProperty r2, r3-r6, [13]
         0x26801c4b026b @   57 : 0d             LdaUndefined
  156 S> 0x26801c4b026c @   58 : a4             Return
```

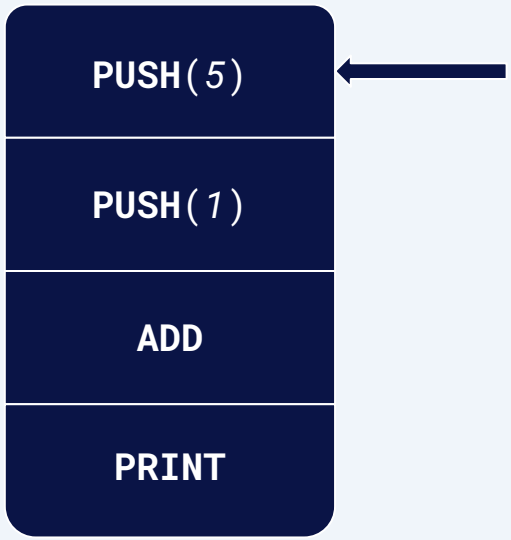
EJECUTANDO PROGRAMAS: BYTECODES

```
2           0 LOAD_GLOBAL              0 (input)
            2 LOAD_CONST               1 ('Escribí un número: ')
            4 CALL_FUNCTION            1
            6 STORE_FAST               0 (num_texto)

3           8 LOAD_GLOBAL              1 (int)
           10 LOAD_FAST                0 (num_texto)
           12 CALL_FUNCTION            1
           14 STORE_FAST               1 (num)

4          16 LOAD_GLOBAL              2 (print)
           18 LOAD_FAST                1 (num)
           20 LOAD_CONST               2 ('al cuadrado es')
           22 LOAD_FAST                1 (num)
           24 LOAD_FAST                1 (num)
           26 BINARY_MULTIPLY
           28 CALL_FUNCTION            3
           30 POP_TOP
           32 LOAD_CONST               0 (None)
           34 RETURN_VALUE
```

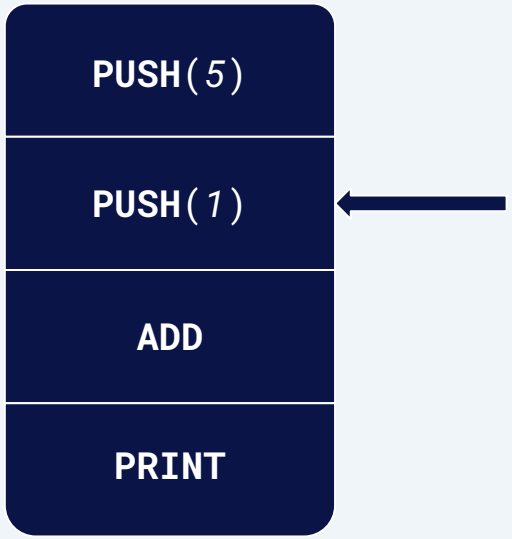
EJECUTANDO PROGRAMAS: MÁQUINAS DE PILA



FIN PILA

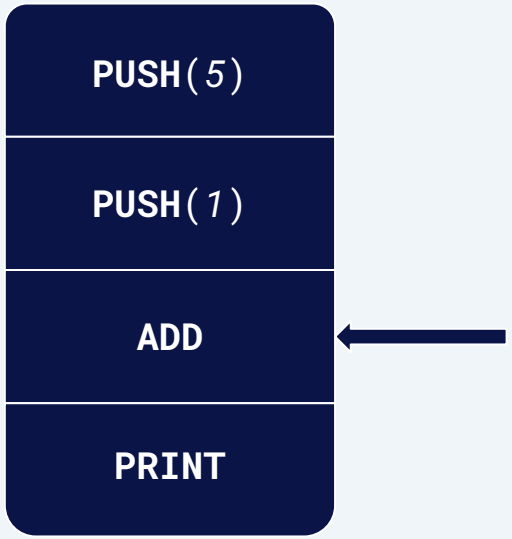
```
~/ECI2021$ ./interprete
ejemplo.ona
Output:
```

EJECUTANDO PROGRAMAS: MÁQUINAS DE PILA



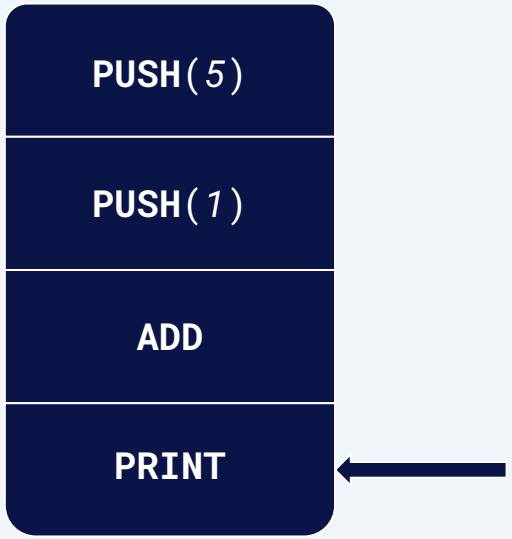

```
~/ECI2021$ ./interprete
ejemplo.ona
Output:
```

EJECUTANDO PROGRAMAS: MÁQUINAS DE PILA




```
~/ECI2021$ ./interprete
ejemplo.ona
Output:
```

EJECUTANDO PROGRAMAS: MÁQUINAS DE PILA




```
~/ECI2021$ ./interprete
ejemplo.ona
Output:
```

EJECUTANDO PROGRAMAS: MÁQUINAS DE PILA


```
~/ECI2021$ ./interprete
ejemplo.ona
Output:
6
```

Muchas gracias!!

THANK YOU

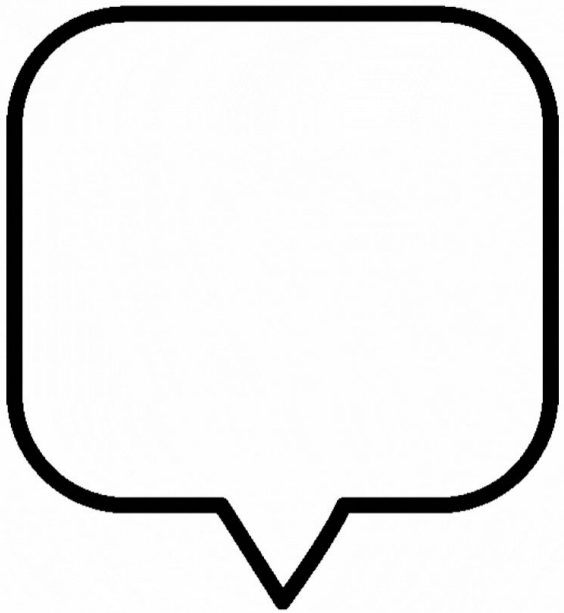
Querés ser parte de nuestra Tribu?

https://www.onapsis.com/company/careers

ONAPSIS.COM

ONAPSIS

PREGUNTAS



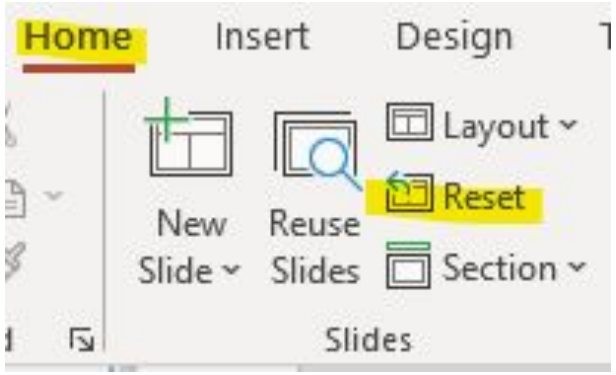
CREATING A NEW PPT | BEST PRACTICES

To avoid copying styles from other presentations and keeping the Onapsis format, please follow these steps:

Paste the text from another source.

Then click “Reset” to delete all text styling. Do this in every single slide.

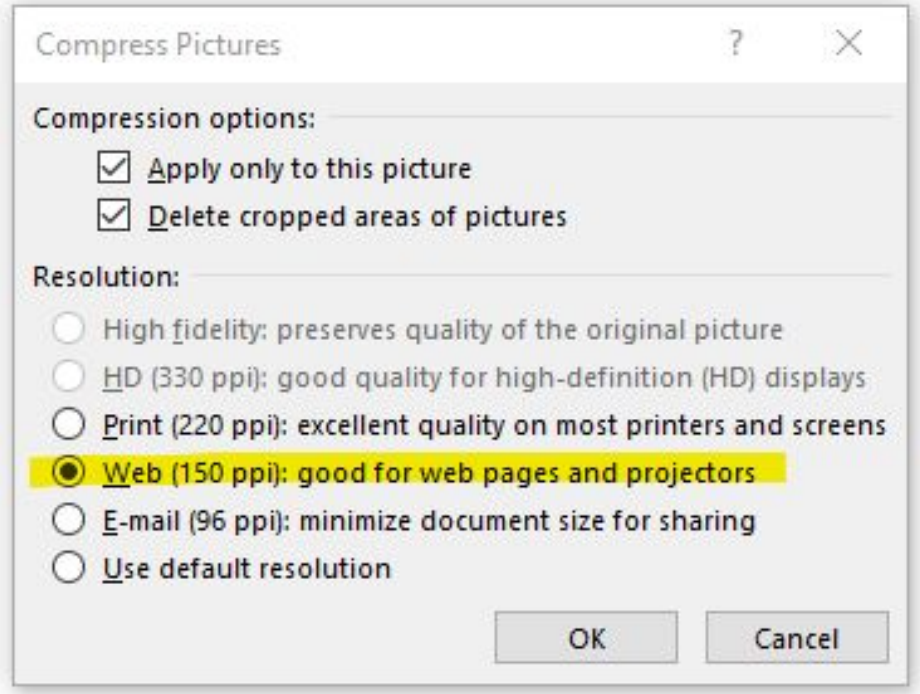
Make sure you do this every time you copy and paste so in that way the Onapsis brand will be maintained.



CREATING A NEW PPT | BEST PRACTICES

To keep your presentation in the small file size possible without losing image quality, please follow these steps:

- Insert an image
- Double click on the image
- Under “Picture Format” click 



GENERAL BRANDING | LOGOS

GENERAL BRANDING | COLORS

Pantone 282 C
CMYK: 100 / 97 / 36 / 47
RGB: 10 / 20 / 70
HEX #0a1446

Pantone 1585 C
CMYK: 0 / 70 / 97 / 0
RGB: 245 / 115 / 35
HEX #f57323

Pantone 2726 C
CMYK: 78 / 79 / 0 / 0
RGB: 90 / 25 / 235
HEX #5a19eb

Pantone 7689 C
CMYK: 79 / 33 / 17 / 0
RGB: 40 / 140 / 180
HEX #288cb4

Pantone 656 C
CMYK: 4 / 1 / 0 / 0
RGB: 240 / 245 / 250
HEX #f0f5fa

ELEMENTS |
ICONS

THIS IS A TRANSITION SLIDE

ANOTHER TEXT GOES HERE