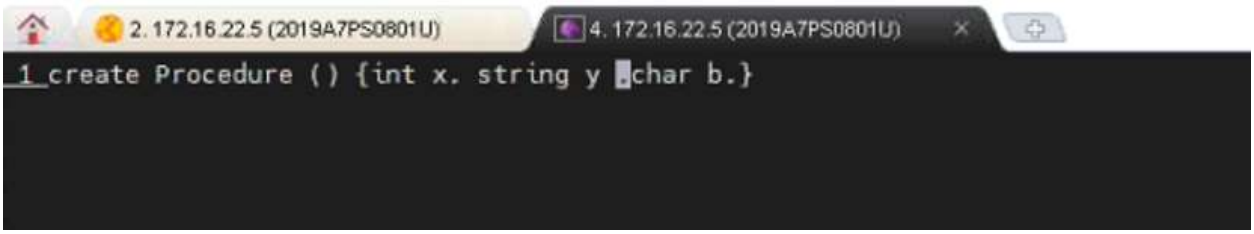
(2) EZ INPUT OUTPUT SNIPPETS

Ez LANGUAGE

INPUT OUTPUT
INPUT – EZ LANGUAGE translated to
OUTPUT – PYTHON (target language)

Python uses indentation, but for the sake of simplicity (visually and programming) we have **used curly brackets to differentiate the blocks in our target program**, assuming that the curly brackets will be replaced by proper indentation.

DECLARATION



output of inputez (**declaration - python** internally sets the data type only when initialized shown in next step so hence the comment is printed)

INITIALIZATION

Initialization In Ez

```
1 create Procedure () {int x. x = 10.}
2
```

Initialization In Python

```
[2019A7PS0801U@linuxbpdc1 Assignment]$ ./mini-compiler < input5ez.txt
def main(): {
        #int internally declared  x
        x=10
 }
[2019A7PS0801U@linuxbpdc1 Assignment]$
```

PRINTING

printing in *ez* uses write with $ to print the value and simply write() for printing the string

```
2. 172.16.22.5 (2019A7PS0801U)    3. /home/mobaxterm
1 create Procedure () {int x. x = 10. Write($x).}
```

Printing In Python

For **printing** value of variable in python, we assign the value and then print using print()

```
[2019A7PS0801U@linuxbpdc1 Assignment]$ ./mini-compiler < input6ez.txt
def main(): {
        #int internally declared  x
        x=10
print( x )
 }
```

READING

Reading In ***Ez*** Done Using Typ()

```
1 create Procedure () {int x. typ(x).}
~
~
```

```
1 create Procedure () {int x. typ(x). char c. typ(c). float y. typ(y).}
```

Reading In Python Done Using Input()

```
[2019A7PS0801U@linuxbpdc1 Assign]$ ./mini-compiler < input7ez.txt
def main(): {
        #int internally declared  x
         x =int(input())

 }
[2019A7PS0801U@linuxbpdc1 Assign]$
```

```
"input7ez.txt" 1L, 70C written
[2019A7PS0801U@linuxbpdc1 Assign]$ ./mini-compiler < input7ez.txt
def main(): {
        #int internally declared  x
         x =int(input())
        #char internally declared c
         c =input() #char is also considered as single length string
        #float internally declared y
         y =float(input())

 }
[2019A7PS0801U@linuxbpdc1 Assign]$
```

CONDITIONAL STATEMENTS

If-Else In *Ez*

create Procedure () { int x. x =12. int y. y=13. if (x==12) { Write($x). } orelse (x==12) { Write($x).} other { Write($y). } }

```
create Procedure () { int x. x =12. int y. y=13. if (x==12) { Write($x). } orelse (x==12) { Write($x).} other { Write($y). } }
```

If-Else in Python

```
[2019A7PS0801U@linuxbpdc1 Assign]$ ./mini-compiler < input8ez.txt
def main(): {
        #int internally declared  x
        x=12
        #int internally declared  y
        y=13
            if x==12:
 {
                print( x )
 }
        elif x==12:
 {
                print( x )
 }
        else :
  {
                print( y )
 }

 }
```

While Loop In *Ez*

```
create Procedure () {int x, x=1. While ( x < 10 ) { Write($x), x = x + 1. } }
```

While Loop in Python

```
[2019A7PS0801U@linuxbpdc1 Assign]$ ./mini-compiler < input9ez.txt
def main(): {
        #int internally declared  x
        x=1
        while x<10:
 {
                print( x )
                x=x+1
        }

 }
[2019A7PS0801U@linuxbpdc1 Assign]$
```

BOX FOR STRINGS (CORRESPONDING TO LISTS IN PYTHON)

BOX/LIST IN EZ

"Box" keyword is used to create lists in ez language and we know that variables without "$" means they're string and not the value of the variable. So , the images show box examples in ez and corresponding translation in python.

```
1 create Procedure () {string box1 = { x, y, z}.}
```

LIST IN PYTHON

```
[2019A7PS0801U@linuxbpdc1 A3]$ ./mini-compiler < input10ez.txt
def main(): {
        #string internally declared

  box1 =[ " x " , " y " , " z " ]
 }
[2019A7PS0801U@linuxbpdc1 A3]$
```

baXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

“@” BUILT IN FUNCTION

The built-in ez function “@” is used to reverse an integer. Python converts integer to string and then uses slicing operation to reverse as show in example. Hence, “@” translates to the python program as shown.

EG1: IN EZ

```
1 create Procedure () { @1234. }
```

EG1: IN PYTHON:

```
[2019A7PS0801U@linuxbpdc1 A3]$ ./mini-compiler < input11ez.txt
def main(): {
        print( str( 1234 ) [::-1]
 #reverse function is built-in in ez, whereas in python it is as given that is it uses slicing
 }
[2019A7PS0801U@linuxbpdc1 A3]$
```

EG2 IN EZ:

```
1 create Procedure () {@714. @1212. @23456. @1417.}
```

EG2 IN PYTHON:

```
[2019A7PS0801U@linuxbpdc1 A3]$ ./mini-compiler < input11ez.txt
def main(): {
        print( str( 714 ) [::-1]
 #reverse function is built-in in ez, whereas in python it is as given that is it uses slicing
        print( str( 1212 ) [::-1]
 #reverse function is built-in in ez, whereas in python it is as given that is it uses slicing
        print( str( 23456 ) [::-1]
 #reverse function is built-in in ez, whereas in python it is as given that is it uses slicing
        print( str( 1417 ) [::-1]
 #reverse function is built-in in ez, whereas in python it is as given that is it uses slicing

 }
[2019A7PS0801U@linuxbpdc1 A3]$
```

OPERATORS:

“x” can’t variable can’t be used since we use it for xor operation.

In Ez

```
create Procedure() {

int a.
a=10.
int b.
b=7.
Write($a^b).
Write($a(x)b).
Write($a->1).
Write($b<-1).
Write($a||b).
Write($a&&b).
}
~
~
~
~
~
```

In Python

```
      |                                                                  ^
~$ ./mini-compiler < Ezoperators.txt.txt
def main(): {
         " a " #int internally declared  a
        a=10
         " b " #int internally declared  b
        b=7
        a**b
print( b )
        a^b
print( b )
        a>>
print( a )
        b<<
print( b )
        a|b
print( b )
        a&b
print( b )

 }
~$
```

CRYPTOGRAPHIC (BUILT-IN FUNCTION)

ENCRYPTION

IN EZ

```
create Procedure() {
int s.
s=340.
encrypt ( s , 1 ).
}
~
~
~
~
~
~
~
~
```

IN PYTHON

```
~$ ./mini-compiler < enc.txt
def main(): {
        #int internally declared  s
        s=340
        def encrypt (s ,1) : { number1 = s + 1  } { number 2 = (str(number1) [::1]) }
 }
~$
```

DECRYPTION

IN EZ

```
create Procedure() {
int s.
s=340.
decrypt ( s , 1 ).
}
~
~
~
~
~
~
~
~
```

IN PYTHON

```
~$ ./mini-compiler < dec.txt
def main(): {
          #int internally declared  s
        s=340
        def encrypt (s ,1) : { number2 = (str( s ) [::1]) } {number1= int(number2) -1
 }
~$
```

SAMPLE INPUT FILE – Covering few of the above programs in 1 function

EZ

```
create Procedure(){
int r.
int s.
s=0.
int t.
typ(t).
While(t>0)
{
r=t%10.
s=s+r.
t=t/10.
}
Write($s).

if (s < 102)
{
Write($s).
}
orelse (s > 102)
{
Write($t).
}
other
{
Write($r).
}

@00111331221.

}

~
```

PYTHON

```
~$ ./mini-compiler < testinput.txt
def main(): {
         " r " #int internally declared  r
         " s " #int internally declared  s
        s=0
         " t " #int internally declared  t
         t  = int(input())
        while t>0:
 {
                r=t10
                s=s+r
                t=t/10
        }

print( s )
            if s<102:
 {

print( s )

 }
        elif s>102:
 {

print( t )

 }
        else :
  {

print( r )

 }
        print( str( 00111331221 )  [::-1]
 #reverse function is built-in in ez, whereas in python it is as given that is it uses slicing

 }
~$ ./mini-compiler < testinput.txt
```