

Introducción a la programación con Python

Clonación

Recursión/manejo de errores

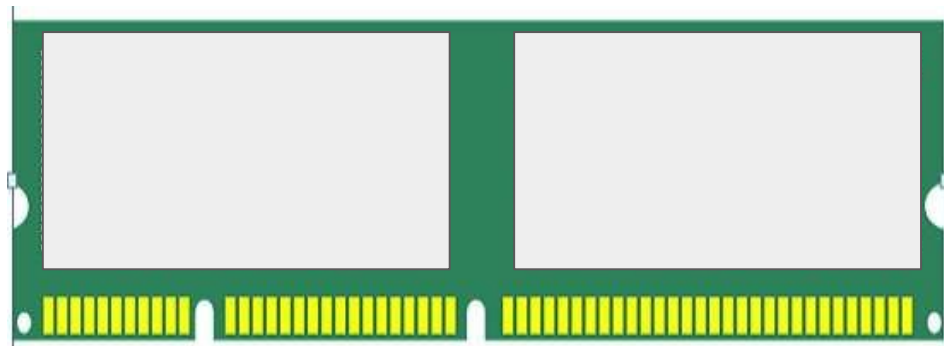
Alexis Rodríguez

Marcel Morán C

Esquema

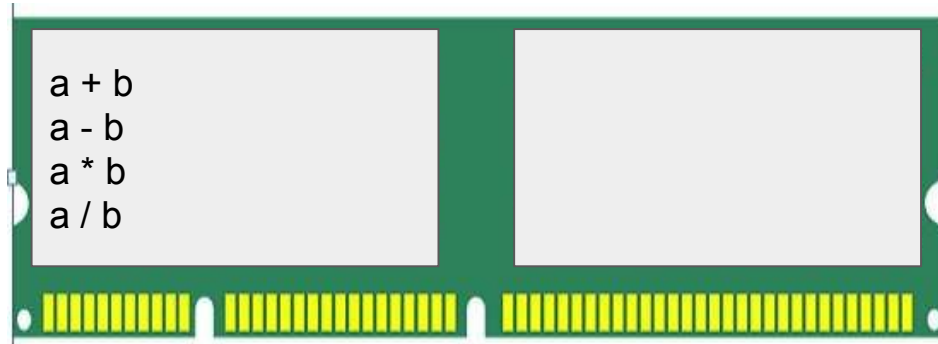
- La memoria
- Clonacion
- ¿Qué es recursión?
- ¿Qué es manejo de errores?
- Sintaxis de manejo de errores
- Tipo de Excepciones

La memoria



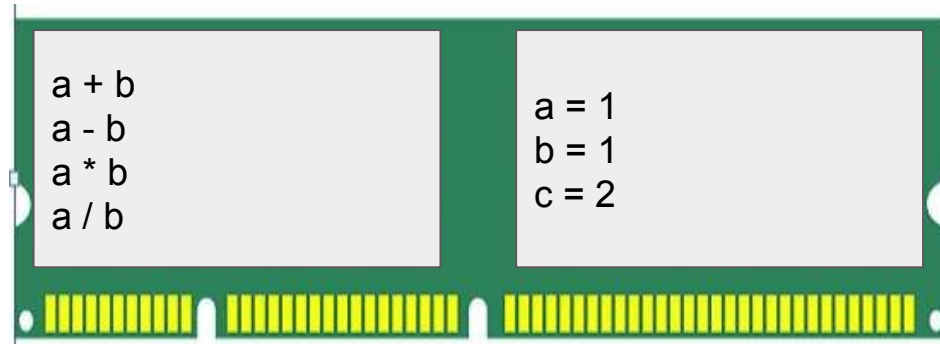
La memoria

- Instrucciones



La memoria

- Instrucciones
- **Tipo de datos**

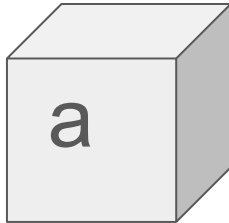


La memoria

Ubicación de memoria	Valores
0x0000	0
0x0001	101010100
0x0002	0
0x0003	100010100

La memoria

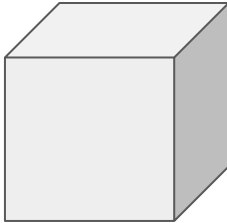
“DATO”



Ubicación de memoria	Valores
0x0000	0
0x0001	101010100
0x0002	0
0x0003	100010100

La memoria

“DATO”



Ubicación de memoria	Valores
0x0000	“DATO”
0x0001	101010100
0x0002	(0x000)
0x0003	100010100

Clonación

- Estructura de tipo de datos pueden ser clonados
- Variables se asignan al mismo de estructura de dato (alias)

x = 10

y = x



Variables	
x	10
y	

Clonación

- Estructura de tipo de datos pueden ser clonados
- Variables se asignan al mismo de estructura de dato (alias)

`x = 10`

`y = x`

`y = y + 1`

Variables	
x	10
y	11

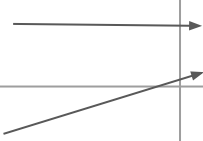
Clonación

- Estructura de tipo de datos pueden ser clonados
- Variables se asignan al mismo de estructura de dato (alias)

x = [1 , 2, 3]

y = x

Variables	
x	[1,2,3]
y	



Clonación

- Estructura de tipo de datos pueden ser clonados
- Variables se asignan al mismo de estructura de dato (alias)

```
x = [1 , 2, 3]
```

```
y = x
```

```
y.append(4)
```

Variables	
x	[1,2,3,4]
y	

Clonación

- Estructura de tipo de datos pueden ser clonados
- Variables se asignan al mismo de estructura de dato (alias)
- **copy()**

```
x = [1 , 2, 3]
```

```
y = x.copy()
```

Variables	
x	→ [1,2,3]
y	→ .[1,2,3]

Clonación

- Estructura de tipo de datos pueden ser clonados
- Variables se asignan al mismo de estructura de dato (alias)
- **copy()**

```
x = [1 , 2, 3]
```

```
y = x.copy()
```

```
y.append(4)
```

Variables	
x	→ [1,2,3]
y	→ [1,2,3,4]

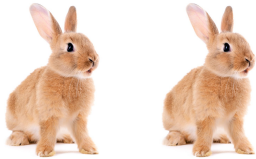
¿Que es recursión?

- Es una manera de resolver un problema usando una función que se llama a sí misma
- Por ejemplo, La secuencia de Fibonacci



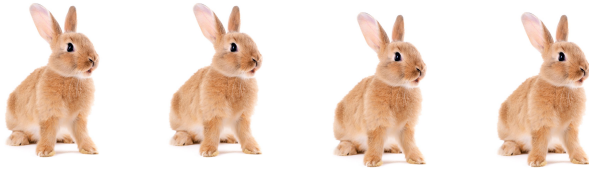
¿Que es recursión?

- Es una manera de resolver un problema usando una función que se llama a sí misma
- Por ejemplo, La secuencia de Fibonacci



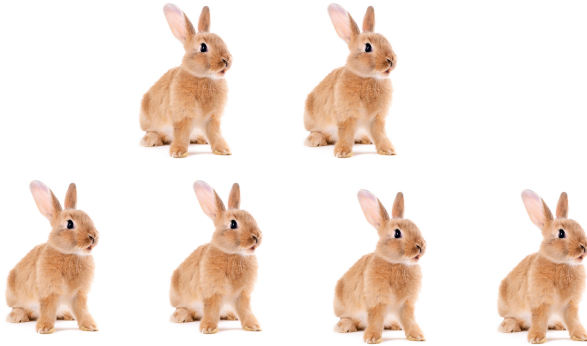
¿Que es recursión?

- Es una manera de resolver un problema usando una función que se llama a sí misma
- Por ejemplo, La secuencia de Fibonacci
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34



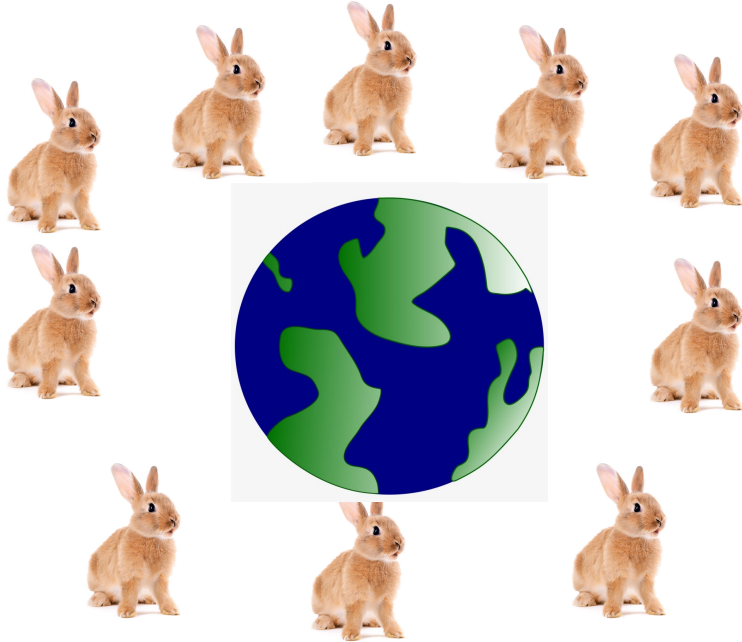
¿Que es recursión?

- Es una manera de resolver un problema usando una función que se llama a sí misma
- Por ejemplo, La secuencia de Fibonacci
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34



¿Que es recursión?

- Es una manera de resolver un problema usando una función que se llama a sí misma
- Por ejemplo, La secuencia de Fibonacci
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34



¿Que es recursión?

- Es una manera de resolver un problema usando una función que se llama a sí misma
- Por ejemplo, La secuencia de Fibonacci
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34
- Caso base
- Caso $n+1$

$$f(0) = 0$$

$$f(1) = 1$$

¿Que es recursión?

- Es una manera de resolver un problema usando una función que se llama a sí misma
- Por ejemplo, La secuencia de Fibonacci
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34
- Caso base
- Caso $n+1$

```
def fibonacci_seq(termino):
```

```
    if(termino <= 1):
```

```
        return termino
```

```
fibonacci_seq(0)
```

```
fibonacci_seq(1)
```

Entrada	Salida
0	0
1	1

¿Que es recursión?

- Es una manera de resolver un problema usando una función que se llama a sí misma
- Por ejemplo, La secuencia de Fibonacci
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34
- Caso base
- Caso n+1

```
def fibonacci_seq(termino):
```

```
    if(termino <= 1):
```

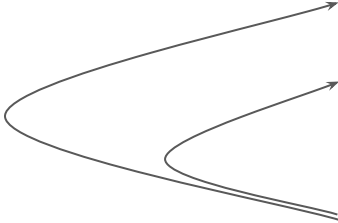
```
        return termino
```

```
    else:
```

```
        return fibonacci_seq(termino-1) + fibonacci_seq(termino-2)
```

```
fibonacci_seq(2)
```

```
fibonacci_seq(3)
```



Entrada	Salida
0	0
1	1
2	1
3	

¿Que es recursión?

- Es una manera de resolver un problema usando una función que se llama a sí misma
- Por ejemplo, La secuencia de Fibonacci
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34
- Caso base
- Caso n+1

```
def fibonacci_seq(termino):
```

```
    if(termino <= 1):
```

```
        return termino
```

```
    else:
```

```
        return fibonacci_seq(termino-1) + fibonacci_seq(termino-2)
```

```
fibonacci_seq(2)
```

```
fibonacci_seq(3)
```

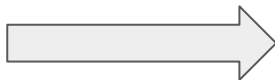
Entrada	Salida
0	0
1	1
2	1
3	2



¿Qué es manejo de errores?

```
def division_ejemplo(numero1, numero2):  
    resultado = numero1 / numero2  
    print('Resultado de la division es:', resultado)
```

division_ejemplo(4, 2)



Resultado de la division es: 2.0

division_ejemplo(4, 0)

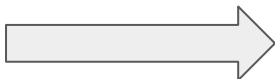


```
ZeroDivisionError                                Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_24784\1793463972.py in <module>  
3     print('Resultado de la division es:', resultado)  
4  
----> 5 division_ejemplo(2, 0)  
~\AppData\Local\Temp\ipykernel_24784\1793463972.py in division_ejemplo(numero1, numero2)  
1 def division_ejemplo(numero1, numero2):  
----> 2     resultado = numero1 / numero2  
3     print('Resultado de la division es:', resultado)  
4  
5 division_ejemplo(4, 0)  
ZeroDivisionError: division by zero
```


Sintaxis de manejo de errores

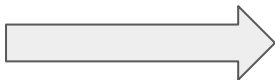
```
def division_ejemplo(numero1, numero2):  
    try:  
        resultado = numero1 / numero2  
        print('Resultado es:', resultado)  
    except ZeroDivisionError:  
        print('Error: No es posible dividir para zero')
```

division_ejemplo(4, 2)



Resultado de la division es: 2.0

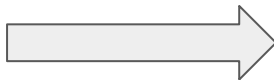
division_ejemplo(4, 0)



Error: No es posible dividir para zero

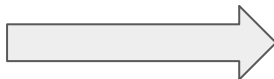
Tipos de excepciones

```
mistring = '45'  
minumero = int(mistring)  
print('mi numero es:', minumero)
```



mi numero es: 45

```
mi string = 'hello'  
mi numero = int(mi string)  
print('mi numero es:', mi_numero)
```



ValueError

Traceback (most recent call last)

```
~\AppData\Local\Temp\ipykernel_24784\2402285655  
.py in <module>  
      1 mi string = 'hello'  
----> 2 mi_numero = int(mi_string)  
      3 print('mi numero es:', mi_numero)
```

ValueError: invalid literal for int() with
base 10: 'hello'

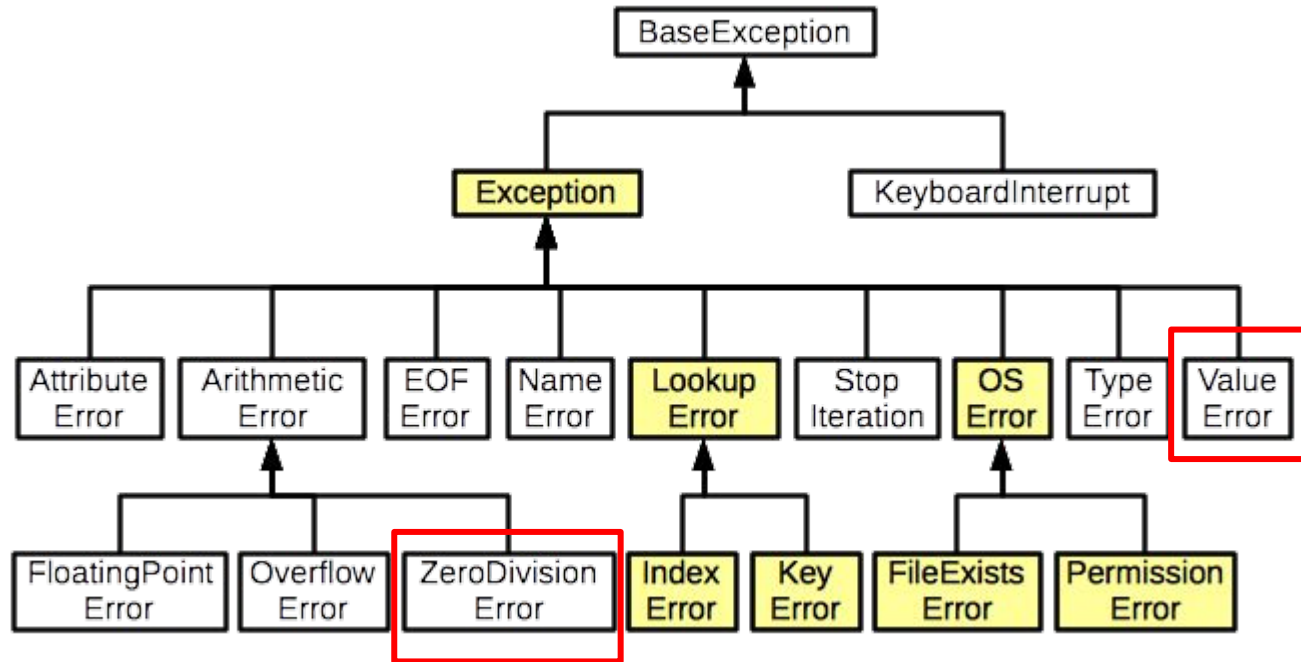
Tipos de excepciones

```
mi_string = 'hello'
try:
    mi_numero = int(mi_string)
    print('mi numero es:', mi_numero)
except ValueError:
    print('Error: Valor invalido', "" + mi_string + "")
```



Error: Valor invalido 'hello'

Tipos de excepciones



Tipos de excepciones

```
try:
```

```
...
```

```
except PrimeraExcepcion:  
    manejar_primera()
```

```
except SegundaExcepcion:  
    manejar_segunda()
```

```
except (TerceraExcepcion, CuartaExcepcion, QuintaExcepcion) as e:  
    manejar_3ra_4ta_5ta()
```

```
except Exception:  
    manejar_todo_lo_demas()
```

← Incluye a todos los
tipos de excepciones

Conclusión

- La memoria almacena datos e instrucciones
- La diferencia de entre la asignación de tipos de datos y tipos de estructura de datos
- Como clonar correctamente la estructura para evitar problemas de asignación
- Recursión utiliza llamadas a una misma función para resolver problemas
- Cómo manejar errores
- Sintaxis de excepciones (try, except) y los tipos que existen