

Informática

Unidad 2I: Lenguajes Interpretados Dinámicos

Introducción a Python

Ingeniería en Mecatrónica

Facultad de Ingeniería
Universidad Nacional de Cuyo



UNIVERSIDAD
NACIONAL DE CUYO



FACULTAD DE INGENIERIA
en acción continua...



C vs Python – Una comparación simplificada



- Apareció en 1972
- Compilado
- Tipado estático
- Paradigma imperativo estructurado
- Gestión de memoria manual
- Usa llaves para delimitar bloques
- Mejor preparado para desarrollos más cercano al dispositivo (menor abstracción)



- Apareció en 1991
- Interpretado
- Tipado dinámico
- Multiparadigma, diseñado como Orientado a Objetos
- Gestión de memoria automática
- Usa indentado como delimitador de bloques
- Mejor preparado para problemas de mayor abstracción



C vs Python - Ejemplo



```
#include <stdio.h>
```

```
//maximo comun divisor
```

```
int mcd(int a, int b) {  
    if (b == 0) {  
        return a;  
    } else {  
        return mcd(b, a % b);  
    }  
}
```

```
int main() {  
    int a,b,c;  
    printf("Ingrese 2 enteros\n");  
    scanf("%d %d", &a, &b);  
    c = mcd(a, b);  
    printf("MCD: %d\n",c);  
    return 0;  
}
```



```
#maximo comun divisor
```

```
def mcd(a, b):  
    if b == 0:  
        return a  
    else:  
        return mcd(b, a % b)
```

```
a = int(input("1er entero: "))  
b = int(input("2do entero: "))  
c = mcd(a, b)  
print("MCD:", c)
```



¿que diferencias y similitudes detectan?

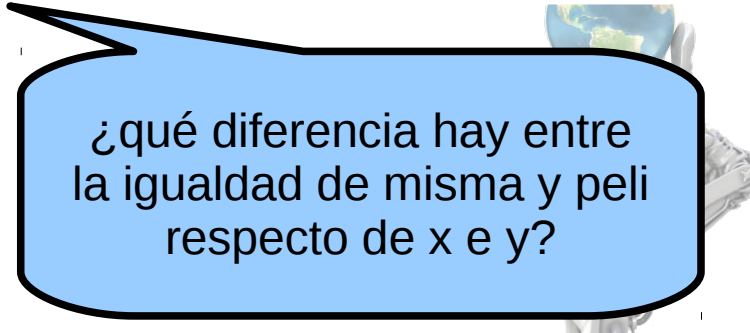
Variables, constantes y tipos

- Las variables se definen sin tipo
- No se puede declarar constantes nombradas
- Los tipos se dividen en mutables e inmutables
 - Regla general: “los tipos primitivos son inmutables; los tipos complejos son mutables”
 - No debe confundirse inmutabilidad con las constantes nombradas
 - Conocer esta propiedad es esencial para entender el paso de parámetros en subrutinas

```
x = 200
print(x, "id", id(x)) # 200 id VAL
y = x
print(y, "id", id(y)) #x e y comparten el mismo valor
x += 1
print(x, "id", id(x)) # id diferente: inmutable!
print(id(x)==id(y)) # True or False?

cadena = 'Hola mundo'
print(cadena[0]) # H
cadena[7] = "M" # Ok or Error?

peli = {'titulo': "Ran", 'año': '?'}
print(peli, 'id', id(peli))
peli['año'] = 1985
print(peli, 'id', id(peli)) #mismo id: mutable!
misma=peli
misma['director'] = "Kurosawa"
print(misma, 'id', id(misma)) #mismo id: mutable!
print(id(misma)==id(peli)) # True or False?1
```

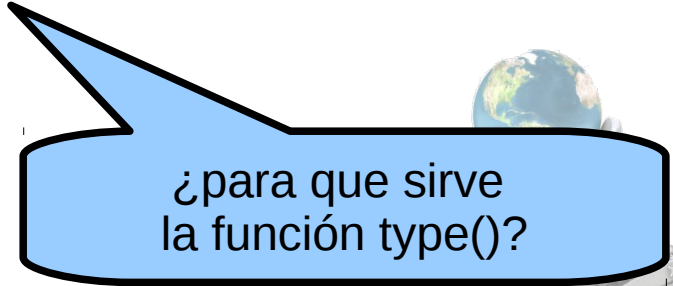


¿qué diferencia hay entre la igualdad de misma y peli respecto de x e y?

Tipos de datos básicos

- Numéricos
 - Autodimensionamiento (no más int, long, float, double, etc)
- Cadenas
 - Se encierran entre comillas simples o dobles.
 - No existe el tipo “char”. Es simplemente una cadena de un caracter.
- Booleanos
 - Ciudadanos de primera clase
 - Se proveen constantes True y False
 - Todo tipo de dato puede transformarse en booleano. La regla general “el vacío/nulo es falso; todo lo demás es verdadero”

```
#ejemplos con tipos de datos
entero = 5
print(type(entero), "y", type(5.0))
cadena = "Hola Mundo"
print('Primer letra:', cadena[0])
z = True
print(type(z), "y es", z)
z = bool(45 and '')
print(type(z), "y ahora es", z)
```



¿para que sirve
la función type()?

Operadores

- Matemáticos, bitwise y relacionales son iguales
 - Python incluye la división entera (//) y la potencia (**) como operaciones nativas
- Los operadores lógicos son idiomáticos (and, or, not)
- La precedencia de operadores es muy similar a C (pero no idéntica)

¿de qué tipo son las variables x e y?

#ejemplos de operadores

```
x = 9
y = 2
print(x/y, "!=" , x//y)
x = x < y**2
print("x < y^2?", x) #imprime?
y = not x
print(x and y, x or y)
```



Condicionales

- Forma general

if expresión:
*a ejecutar si expresión
es verdadera*

elif expresión2:
*a ejecutar si expresión2
es verdadera*

else:
*a ejecutar si nada de lo
anterior es verdadero*

- No olvidar los : (dos puntos) y la indentación correcta para anidar
- Python no provee switch

```
#ej de condicional múltiple
num = float(input("Ingrese un real: "))
if num == 0:
    print("Es cero")
elif num > 0:
    print("Es positivo")
else:
    print("Es negativo")

#ej de condicional anidado
sig = int(num)
if sig == num:
    print("El valor es también entero")
else:
    if num - sig > 0.5:
        sig = sig + 1
    print(sig, "es el redondeo significativo")
```

¿que pasaría si el print()
tuviese una indentación menor?



Bucles

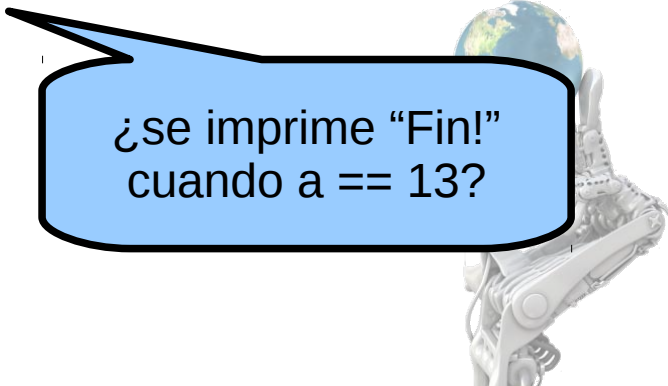
- Bucle “while” es igual al bucle en C

while *expresión:*

*a ejecutar mientras
expresión sea
verdadera*

- Bucle “for” es totalmente diferente al de C (lo veremos más adelante)
- Los bucles admiten la cláusula “else:” que se ejecuta cuando no se cumple la condición
- No posee bucle “do while”
- Al igual que C posee break y continue con idéntico propósito

```
#programa cabulero que imprime impares
a = int(input("Entero desde: "))
b = int(input("Entero hasta: "))
while a <= b :
    if a % 2 != 0 :
        print(a, ', ', end='', sep='')
    a += 1
    if a == 13:
        break
else:
    print("Fin!")
```



¿se imprime “Fin!”
cuando a == 13?

Subrutinas y modularización

- El calificativo “def” permite declarar métodos y funciones:

def *nombre(parámetros):*
codigo de la subrutina

- Los parámetros se pasan “por referencia”, salvo para los tipos de datos inmutables que se comportan por valor.
- Opcionalmente se puede identificar la cláusula *return* para devolver un valor
- La cláusula *import* permite importar módulos y paquetes externos.
- Un módulo es simplemente un archivo .py accesible desde el programa que se importa
- Python provee funciones “built-in” que se utilizan directamente



ejmod.py

```
"""
Módulo para resolver el Máximo común divisor
"""

mcdTit = 'Máximo común divisor: '

def mcd(a, b):
    if b == 0:
        return a
    else:
        return mcd(b, a % b)
```



mcd.py

```
import ejmod

print("Documentación:", ejmod.__doc__)

a = int(input("1er entero: "))
b = int(input("2do entero: "))
print(ejmod.mcdTit + str(ejmod.mcd(a, b)))
```

Estructuras de datos

- No posee “struct” como C.
 - La evolución natural de los lenguajes orientado a objetos se llaman clases
- Tampoco posee arreglos de tipos nativos.
- Posee un conjunto nativo de estructuras de datos.
 - list: array de tamaño variable.
 - tuple: list, pero inmutable
 - set: list, pero no admite duplicados
 - dict: listas indexadas por claves

```
def print_peli(peli):  
    if 'título' in peli:  
        print("Título:", peli['título'])  
    if 'año' in peli:  
        print("Año:", peli['año'])  
  
def definir_año(peli, año):  
    peli['año'] = año  
  
peli = {'título': 'Pulp Fiction'}  
definir_año(peli, 1994)  
print_peli(peli)  
  
lista_de_pelis = [peli]  
nueva_peli = {'título': 'Primer'}  
lista_de_pelis.append(nueva_peli)  
  
print('Tamaño:', len(lista_de_pelis))  
for peli in lista_de_pelis:  
    print_peli(peli)  
  
lista_de_pelis.remove(peli)
```

