

Cronograma

- Salida en pantalla
- Estructuras de control
 - If
 - Ciclos predefinidos (for)
 - Ciclos indefinidos (while)
 - Rompiendo ciclos (break)

Codificación



- Podemos definir el tipo de codificación en nuestro modulo.
 - Codificación UTF-8
- PEP 263
- Primera línea de código de cada módulo.
`# -*- coding: utf-8 -*-`

Salida en pantalla



- Salidas con estilo C (el viejo estilo)
 - Sigue vigente en Python 3 pero su uso es cada vez menor.
 - Se recomienda no usar más esta manera (eficiencia)
 - Estilo viejo:
 - Se utiliza el carácter %
 - Se especifica el tipo del valor a imprimir
 - En su lugar se propone utilizar los **f-string**

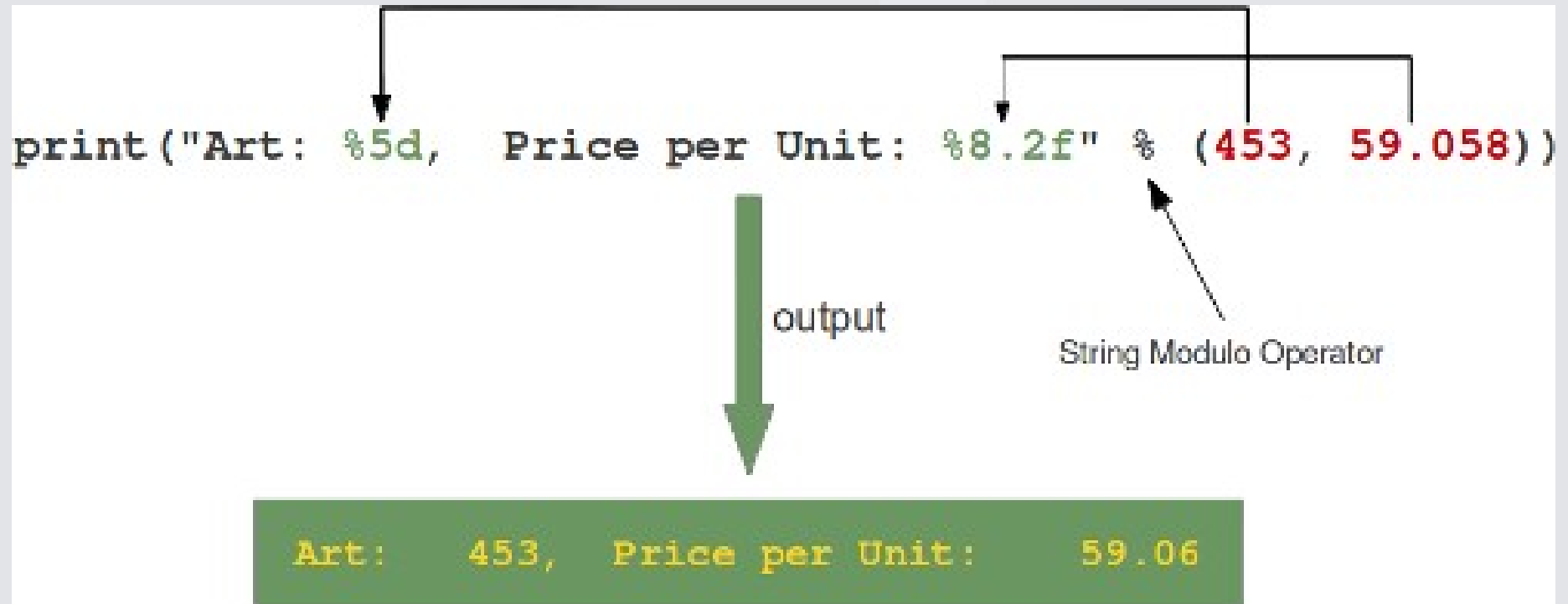
Salida en pantalla (2)

```
print("Art: %5d, Price per Unit: %8.2f" % (453, 59.058))
```

output

String Modulo Operator

```
Art:    453, Price per Unit:    59.06
```



Salida en pantalla (3)

```
# integer
```

```
print("%d" % (1234))
```

```
# espaciado 4 lugares
```

```
print("%4d" % (12))
```

```
# float
```

```
print("%10.4f" % (3.123456789))
```

```
# string and integer
```

```
print("hola que %5s tal %d" % ("cats", 4))
```

Salida en pantalla (4)

- Salidas con formato *str.format()*
 - El texto que queremos formatear puede tener uno o varios campos a reemplazar.
 - *print ('Somos los {0} quienes decimos {1}!' .format('caballeros', 'No'))*
 - *print ("La suma de 1+2 es {0}".format(1+2))*
 - La suma de 1+2 es 3
 - *print (" {0:d} {1:b} {2:x}".format(2,2*3,3*4*5))*
 - Decimal, binario y hexadecimal (octal es con la letra "o")

Salida en pantalla (6)

Old

```
'%s %s' % ('one', 'two')
```

New

```
'{} {}'.format('one', 'two')
```

Output

```
o n e   t w o
```

Old

```
'%d %d' % (1, 2)
```

New

```
'{} {}'.format(1, 2)
```

Output

```
1   2
```

f-Strings

- Nueva y mejorada forma de formatear cadenas en Python
- A partir desde la **versión 3.6**
- Formato: `f{}` o `F{}`
- Los f-String son más eficientes. Las cadenas F son más rápidas que el `%` de formato y `str.format()`.
- Son expresiones evaluadas en tiempo de ejecución en lugar de valores constantes.
- Ejemplo:

```
nombre = "Eric"  
edad = 74  
print(f"Hola, {nombre} . Tienes {edad} .")
```


Estructuras de control

- ***if <condición>:***
 - ***<hacer algo si se da la condición>***
- ***if*** es una palabra reservada y la condición es una expresión de tipo booleana en donde su evaluación sera True o False.

Expresiones de comparación	
Expresión	Significado
a == b	a es igual a b
a != b	a es distinto de b
a < b	a es menor que b
a <= b	a es menor o igual que b
a > b	a es mayor que b
a >= b	a es mayor o igual que b

Operadores Lógicos	
Expresión	Significado
a and b	El resultado es True sii a es True y b es True
a or b	El resultado es True si a es True o b es True
not a	El resultado es True si a es False de lo contrario el resultado es False

Estructuras de control (2)

Algunas formas de los condicionales.

Python no tiene switch-case

```
if <condicion>:  
    accion  
    accion  
    ....  
    accion
```

```
if <condicion>:  
    accion  
    ....  
    accion  
else:  
    accion  
    ....  
    accion
```

```
if <condicion>:  
    accion  
    ....  
    accion  
elif <condicion>:  
    accion  
    ....  
    accion  
....  
else:  
    accion  
    ....  
    accion
```

```
var = "par" if (num % 2 == 0) else "impar"
```

Estructuras de control (3)

- Cualquier estructura de control puede anidarse dentro de cualquier otra.
- Solamente hay que tener cuidado de salir en algún momento de las estructuras. ¡Cuidado con las indentaciones!
- Ejemplo: Decidir si un numero es positivo cero ó negativo no.

```
numero = int(input("Ingrese un numero: "))
if numero > 0:
    print("Numero positivo")
elif numero == 0:
    print("Igual a 0")
else:
    print("Numero negativo")
```

```
numero = int(input("Ingrese un numero: "))
if numero > 0:
    print("Numero positivo")
else:
    if numero == 0:
        print("Igual a 0")
    else:
        print("Numero negativo")
```

Estructuras de control (4)

- En Python se considera como verdadero:
 - Los valores numéricos distintos de 0
 - Las cadenas de caracteres que no son vacías
 - En general cualquier valor que no sea 0 o vacío
- Mientras que los valores 0 o vacíos se consideran falsos

```
num = 3
print (num)
>> 3
num = bool(num)
print (num)
>> True
```

Estructuras de repetición

- Sentencias repetitivas (ciclos)

while condicion:

```
accion  
accion  
.....  
accion
```

for variable **in** serie de valores:

```
accion  
accion  
.....  
accion
```

Ciclos predefinidos

- Podemos recorrer cualquier secuencia de valores y realizar alguna acción sobre ellos. Para esto utilizamos la instrucción **for**
 - **for x in <secuencia de valores>:**
 <hacer algo con x>
- La variable x toma valores enteros en el intervalo generado (y perdura después del ciclo)
- Generalmente se utiliza la función range() para generar una secuencia de valores.

for x in range(n1,n2):
 <hacer algo con x>

Ciclos (2)

- La función *range* genera la secuencia de valores enteros del intervalo $[n1, n2)$
- También podemos indicar una secuencia de valores como:
 - `range(n)`. Establece como secuencia de valores a $[0, 1, \dots, n-1]$.
 - `range(n1, n2, salto)`. Establece como secuencia de valores indicando el salto.

Ciclos (3)

- Que pasa si queremos que el usuario pueda ingresar muchos números y cada vez que se ingresa uno debemos informar si es positivo, cero o negativo
- Una solución es preguntar cuantos números quiere ingresar el usuario y luego preguntar las veces que ingreso.

```
i = int(input("Cuanto numeros quiere procesar: "))
for j in range(i):
    x = int(input("Ingresa un numero: "))
    if x > 0:
        print("Numero positivo")
    elif x == 0:
        print("Numero igual 0")
    else:
        print("Numero negativo")
```


Ciclos (4)

- Pero si no sabemos de antemano cuántos números vamos a ingresar...
- Debemos tener alguna forma de repetir la pregunta hasta que se cumple alguna condición de salida.
- Para esto tenemos la instrucción ***while***
- ***while*** <condición>:
 <hacer algo>

Ciclos (5)

- ***while*** es una palabra reservada, la condición es una expresión booleana, igual que en las instrucciones ***if*** y el cuerpo es, como siempre, una o más instrucciones de Python.
- El sentido de esta instrucción es el siguiente:
 1. Evaluar la condición.
 2. Si la condición es falsa, salir del ciclo.
 3. Si la condición es verdadera, ejecutar el cuerpo.
 4. Volver a 1.

Ciclos (6)

- También podemos seguir preguntando al usuario por un número mientras no ingresa una orden de salida.
- Para esto definimos una variable de corte.

```
continuar_jugando = "Si"
while continuar_jugando == "Si":
    x = int(input("Ingrese un numero: "))
    if x > 0:
        print("Numero positivo")
    elif x==0:
        print("Igual a 0")
    else:
        print("Numero negativo")
    continuar_jugando = input("Quiere seguir? <Si-No>: ")
```

Rompiendo ciclos

- En Python tenemos una instrucción ***break*** que nos permite salir de adentro de un ciclo (tanto sea ***for*** como ***while***) en medio de su ejecución.
- ***while*** <condicion>:
 <hacer algo_1>
 if <condif>:
 break
 <hacer algo_2>

Ejercicios

- **Ejercicio 1**

- Imprimir todos los números pares y su cuadrado del 0 al 100.

- **Ejercicio 2**

- Dada una palabra ingresada por el usuario contar cuantas ocurrencias de la letra “a” tiene.

- **Ejercicio 3**

- Dada una cadena de caracteres:
 - Imprima dicha cadena cada dos caracteres. Ej.: 'recta' debería imprimir 'rca'

Ejercicios

- Ejercicio 4:
 - **Manejo de contraseñas**
 - a) Escribir un programa que contenga una contraseña inventada y que se la pregunte al usuario. No permitirá continuar hasta que haya ingresado correctamente.
 - b) Modificar el programa anterior para que solamente permita una cantidad fija de intentos.
 - c) Modificar el programa anterior para que después de cada intento agregue una pausa cada vez mayor, utilizando la función ***sleep*** del módulo ***time***.
 - import time
 - time.sleep(secs)

Ejercicios (2)

- Ejercicio 5:
 - Utilizando la función ***randrange*** del módulo ***random***, escribir un programa que obtenga un número aleatorio secreto, y luego permita al usuario ingresar números y le indique si son menores o mayores que el número a adivinar, hasta que el usuario ingrese el número correcto.

```
import random
```

```
valor = random.randrange(start, stop[,  
step])
```