

Introducción a Python

Variables

- Una variable es un lugar designado en la memoria donde el programador puede guardar los datos y luego recuperar esos datos utilizando el "nombre" de la variable
- Los programadores elijen los nombres de las variables
- Usted puede cambiar el contenido de una variable en un enunciado posterior

```
x = 12.2
```

x 12.2

```
y = 14
```

y 14

Palabras Reservadas

- No puede utilizar las palabras reservadas como nombres o identificadores de variables

False	class	return	is	finally
None	if	for	lambda	continue
True	def	from	while	nonlocal
and	del	global	not	with
as	elif	try	or	yield
assert	else	import	pass	
break	except	in	raise	

Variables

- Una variable es un lugar designado en la memoria donde el programador puede guardar los datos y luego recuperar esos datos utilizando el "nombre" de la variable
- Los programadores elijen los nombres de las variables
- Usted puede cambiar el contenido de una variable en un enunciado posterior

```
x = 12.2
```

x ~~12.2~~ 100

```
y = 14
```

y 14

```
x = 100
```

Constantes

- Los valores fijos como los números, letras y cadenas reciben el nombre de "constantes" porque su valor no cambia
- Las constantes numéricas son las que usted espera
- Las constantes de la cadena son comillas simples (') o dobles (")

```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hola mundo')
Hola mundo
```

Reglas para el Nombre de Variables en Python

Debe comenzar con una letra o guión bajo_
 Debe constar de letras, números y guión bajo
 Es sensible a la mayúscula y minúscula

Bien:	spam	eggs	spam23	_speed
Mal:	23spam	#sign	var.12	
Diferente:	spam	Spam	SPAM	

Sentencias o Líneas

`x = 2` ← Enunciado de asignación
`x = x + 2` ← Asignación con expresión
`print(x)` ← Función print (imprimir)

Variable Operador Constante Función

Expresiones Numéricas

```

>>> xx = 2
>>> xx = xx + 2
>>> print(xx)
4
>>> yy = 440 * 12
>>> print(yy)
5280
>>> zz = yy / 1000
>>> print(zz)
5.28
  
```

```

>>> jj = 23
>>> kk = jj % 5
>>> print(kk)
3
>>> print(4 ** 3)
64
  
```

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
**	Potencia
%	Resto

Enunciados de Asignación

Asignamos un valor a una variable utilizando el enunciado de asignación (=)

Un enunciado de asignación consta de una expresión en el lado derecho y una variable para almacenar el resultado

`x = 3.9 * x * (1 - x)`

Reglas de Precedencia del Operador

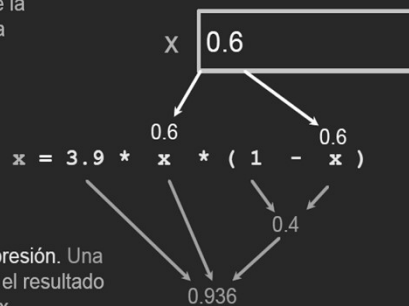
De la regla de precedencia más alta a la regla de precedencia más baja:

- Siempre se respetan los paréntesis
- Potenciación (elevar a la potencia)
- Multiplicación, división, resto
- Suma y resta
- Izquierda a derecha

Paréntesis
 Potencia
 Multiplicación
 Suma
 Izquierda a derecha



Una variable es un lugar de la memoria que se utiliza para guardar un valor (0.6)

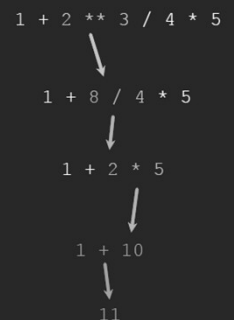


El lado derecho es una expresión. Una vez evaluada la expresión, el resultado se coloca en (se asigna a) `x`.

```

>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
>>>
  
```

Paréntesis
 Potencia
 Multiplicación
 Suma
 Izquierda a derecha



¿Qué Significa “Type” (Tipo)?

- En Python, las variables, literales y constantes tienen un “type” (tipo)
- Python sabe la diferencia entre un número entero y una cadena
- Por ejemplo “+” significa “suma” si se trata de número y “concatenación” si se trata de una cadena

```
>>> ddd = 1 + 4
>>> print(ddd)
5
>>> eee = 'hola ' + 'a
  todos'
>>> print(eee)
Hola a todos
```

concatenación = unión

Conversiones de Type (Tipo)

- Cuando introduces un número entero y un decimal en una expresión, el entero (int) se convierte implícitamente en uno decimal (float)
- Puede controlar esto con las funciones incorporadas int() y float()

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class'float'>
>>>
```

El “Type” (Tipo) Importa

- Python sabe cual es el “type” de todo
- Algunas operaciones están prohibidas
- No se puede “agregar 1” a una cadena
- Podemos preguntarle a Python de qué tipo se trata con la función type()

```
>>> eee = 'hola ' + 'a todos'
>>> eee = eee + 1
Trazas de rastreo (llamada más reciente a lo último): Archivo
"<stdin>", línea 1, in
<module>TypeError: Can't convert
'int' object to str implicitly
>>> type(eee)
<class'str'>
>>> type('hola')
<class'str'>
>>> type(1)
<class'int'>
>>>
```

Input (Entrada) del Usuario

- Podemos instruirle a Python que haga una pausa y lea los datos del usuario con la función input()
- La función input() regresa a la cadena

```
nam = input('Quién es usted')
print('Bienvenido', nam)
```

Quién es usted
Chuck
Bienvenido Chuck

Diferentes Types (Tipos) de Número

- Los números tienen dos types (tipos)
 - Enteros (int):
-14, -2, 0, 1, 100, 401233
 - Números con punto flotante (float), que tienen decimales: -2.5, 0.0, 98.6, 14.0
- Hay otros tipos de números: son variantes entre los números decimales y los números enteros

```
>>> xx = 1
>>> type(xx)
<class'int'>
>>> temp = 98.6
>>> type(temp)
<class'float'>
>>> type(1)
<class'int'>
>>> type(1.0)
<class'float'>
>>>
```

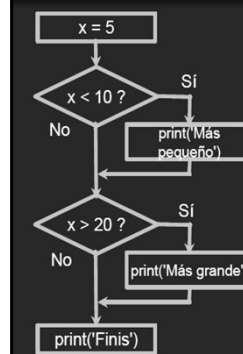
Comentarios en Python

- Todo lo que aparezca luego de # es ignorado por Python
- ¿Por qué usar comentarios?
 - Permiten describir lo que está pasando en la secuencia de un código
 - Permiten documentar quién escribió el código o la información auxiliar
 - Permiten desactivar la línea de un código, quizás de manera temporaria

Scripts de Python

- Interactive Python (Python interactivo) es bueno para los experimentos y programas de 3-4 líneas de largo.
- La mayoría de los programas son mucho más largos, entonces los escribimos en un archivo y le decimos a Python que ejecute los comandos en el archivo.
- De algún modo, le estamos “dando un script (guión) a Python”.
- Como convención, agregamos “.py” como sufijo al final de estos archivos para indicar que contienen Python.

Pasos Condicionales



Programa:

```

x = 5
if x < 10:
    print('Más pequeño')
if x > 20:
    print('Más grande')
print('Finis')
  
```

Resultado:

Más pequeño
Finis

Interactivo versus Script

- Interactivo
 - Usted escribe directamente en Python de a una línea por vez y el programa responde
- Script
 - Usted ingresa una secuencia de enunciados (líneas) en un archivo utilizando un editor de texto y le dice a Python que ejecute los enunciados en el archivo

Operadores de Comparación

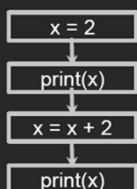
- Las expresiones booleanas formulan una pregunta y generan un resultado Yes (afirmativo) o No (negativo) que utilizamos para controlar el flujo del programa
- Las expresiones booleanas utilizan operadores de comparación para evaluar si es True (Verdadero) / False (Falso) o Yes (Sí) / No
- Los operadores de comparación observan las variables pero no las modifican

Python	Significado
<	Menor que
<=	Menor que o Igual a
==	Igual a
>=	Mayor que o igual a
>	Mayor que
!=	No igual a

Recuerde: “=” se usa para asignación.

http://en.wikipedia.org/wiki/George_Boole

Pasos Secuenciales



Programa:

```

x = 2
print(x)
x = x + 2
print(x)
  
```

Resultado:

2
4

Cuando se está ejecutando un programa, fluye de un paso al otro. Como programadores, configuramos los “paths” (caminos) que el programa debe seguir.

Operadores de Comparación

```

x = 5
if x == 5 :
    print('Igual a 5')
if x > 4 :
    print('Mayor que 4')
if x >= 5 :
    print('Mayor que o Igual a 5')
if x < 6 : print('Menor que 6')
if x <= 5 :
    print('Menor que o Igual a 5')
if x != 6 :
    print('No igual a 6')
  
```

Igual a 5

Mayor que 4

Mayor que o Igual a 5

Menor que 6

Menor que o Igual a 5

No igual a 6

Indentación

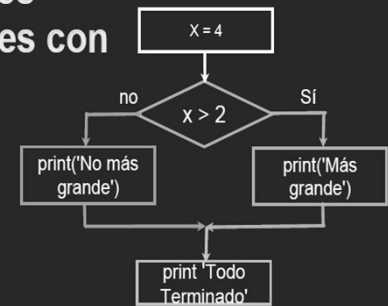
- Aumentar la indentación sirve para indentar luego de un enunciado if o for (después:)
- Mantener la indentación sirve para indicar el alcance del bloque (qué líneas son afectadas por if/for)
- Reducir la indentación permite regresarla al nivel del enunciado if o for para indicar el final del bloque
- Las líneas en blanco son ignoradas y no afectan la indentación
- Los comentarios en una línea en sí mismos se ignoran en lo que respecta a la indentación

Decisiones Bidireccionales con else:

x = 4

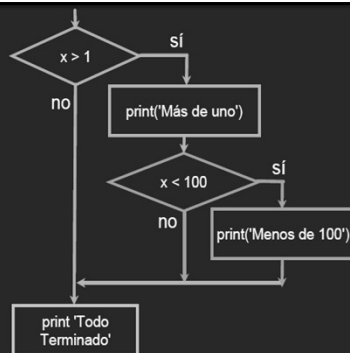
```
if x > 2 :
    print('Más grande')
else :
    print('Más pequeño')
```

print 'Todo Terminado'



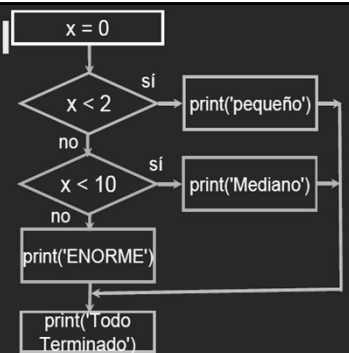
Decisiones Anidadas

```
x = 42
if x > 1 :
    print('Más de 1')
    if x < 100 :
        print('Menos de 100')
print('Todo Terminado')
```



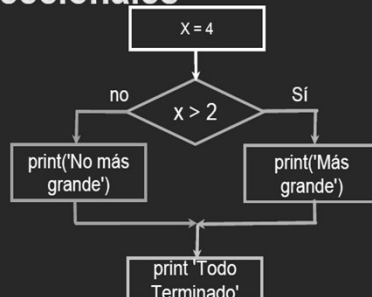
Multidireccional

```
x = 0
if x < 2 :
    print('pequeño')
elif x < 10 :
    print('Mediano')
else :
    print('ENORME')
print('Todo terminado')
```



Decisiones Bidireccionales

- A veces, queremos hacer una cosa si una expresión lógica es verdadera y otra cosa si la expresión es falsa
- Es como una encrucijada – debemos elegir un camino u otro pero no podemos elegir ambos



La Estructura try / except

- Usted rodea una sección peligrosa del código con try y except
- Si el código en try funciona – except es omitido
- Si el código en try falla – pasa a la sección except

Muestra de try / except

```
rawstr = input('Ingresar un número:')
try:
    ival = int(rawstr)
except:
    ival = -1
if ival > 0 :
    print('Buen trabajo')
else:
    print('No es un número')
```

\$ python3 trynum.py
Ingresar un número:42
Buen trabajo
\$ python3 trynum.py
Ingresar un
número:cuarenta-y-dos
No es un número
\$

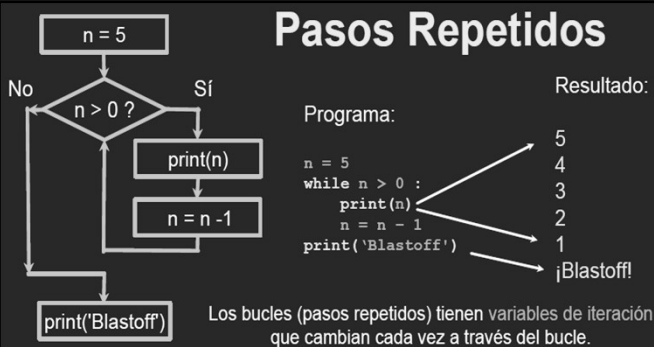
Romper un Bucle

- El enunciado break (romper) termina el bucle actual y salta al enunciado que le sigue inmediatamente al bucle
- Es como una prueba de bucle que puede suceder en cualquier lado en el cuerpo del bucle

```
while True:
    línea = input('> ')
    if línea == 'terminado':
        break
    print(línea)
    print('Terminado')
```

> hola
> hola
> finished
> done
> terminado

Pasos Repetidos



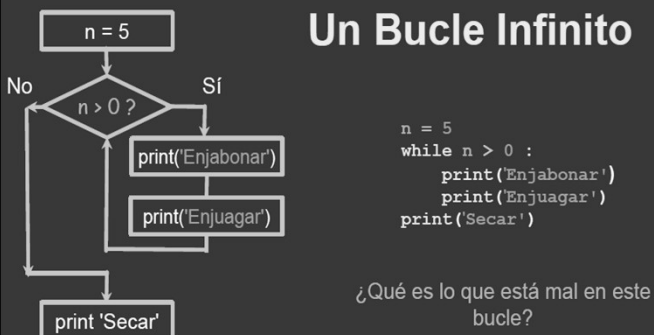
Finalizar una Iteración con Continue

El enunciado continue (continuar) termina la iteración actual y salta a la parte superior del bucle y comienza la siguiente iteración

```
while True:
    línea = input('> ')
    if línea[0] == '#':
        continue
    if línea == 'terminado':
        break
    print(línea)
    print('Terminado')
```

> hola
> hola
> # no imprimir esto
> Imprimir esto
> terminado
> Terminado

Un Bucle Infinito



Bucles Definidos

- Con bastante frecuencia tenemos una lista de los ítems de las líneas en un archivo, es decir un conjunto finito de cosas
- Podemos escribir un bucle para ejecutar el bucle una vez para cada uno de los ítems de un conjunto utilizando la secuencia for de Python
- Estos bucles se denominan "bucles definidos" porque se ejecutan una cantidad exacta de veces
- Decimos que los "bucles definidos" iteran a través de los miembros de un conjunto

Un Bucle Definido Simple

```
for i in [5, 4, 3, 2, 1] :
    print(i)
print('Blastoff')
```

5
4
3
2
1
Blastoff

Un Bucle Definido con Cadenas

```
amigos = ['Joseph', 'Glenn', 'Sally']
for amigo in amigos :
    print('Feliz año nuevo:', amigo)
print('Terminado')
```

Feliz año nuevo: Joseph
Feliz año nuevo: Glenn
Feliz año nuevo: Sally
¡Terminado!

Observando a In...

- La variable de iteración "itera" a través de la secuencia (conjunto ordenado)
- El bloque (cuerpo) del código se ejecuta una vez para cada valor in de la secuencia
- La variable de iteración se mueve a través de todos los valores in de la secuencia

Variable de iteración

Secuencia de cinco elementos

```
for i in [5, 4, 3, 2, 1] :
    print(i)
```

Creando Bucles "inteligentes"

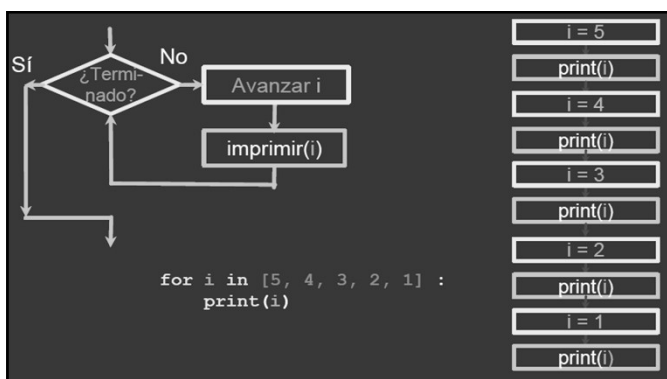
El truco consiste en "conocer" algo acerca del bucle entero cuando está estancado escribiendo código que solo ve una entrada por vez

Configure algunas variables con los valores iniciales

para objeto en los datos:

Buscar o hacer algo para cada entrada por separado, que actualice una variable

Observe las variables




Para encontrar el mayor valor

```
largest_so_far = -1
print('Antes', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)
print('Después', largest_so_far)
```


\$ python largest.py
Antes -1
9 9
41 41
41 12
41 3
74 74
74 15
Después 74

Creamos una variable que contenga el mayor valor que se haya visto hasta ahora (largest_so_far). Si el número actual que estamos buscando es más grande, entonces será el nuevo mayor valor que se haya visto hasta ahora (largest_so_far).

Introducción – Parte 4

PYTHON PARA TODOS 

Agradecimientos / Colaboraciones



Estas diapositivas están protegidas por derechos de autor 2010- Charles R. Severance (www.cse.chadwick.com) de la Facultad de Información de la Universidad de Michigan, y se ponen a disposición bajo licencia de Creative Commons Attribution 4.0. Por favor, conserve esta última diapositiva en todas las copias del documento para cumplir con los requisitos de atribución de la licencia. Si realiza algún cambio, síntese libre de agregar su nombre y el de su organización a la lista de colaboradores en esta página cuando republique los materiales.

Desarrollo inicial: Charles Severance, Facultad de Información de la Universidad de Michigan
... Ingrese nuevos colaboradores y traductores aquí

Continúa ...