Lenguaje de Programación Python

Sentencias

Dr. Mario Marcelo Berón
Argentina Programa
Universidad Nacional de San Luis

Tipos Primitivos y sus Interpretaciones Booleanas

La sentencia *if* permite tomar una decisión dependiendo del valor de una *condición*.
Una *condición* la cual es una expresión que evalúa a *True* o *False*.

Tipos Primitivos y sus Interpretaciones Booleanas

int	0	False
	-1	True
	124	True
float	0.0	False
str		False
	“False”	True
dic		False
	’key’:’val’	True
list	[]	False
	[false]	True

- Todos los tipos primitivos pueden ser usados en las sentencias if
- 0 es falso cuaquier otro valor distinto de 0 es verdadero.
- Contenedores vacíos son falsos

Tipos Primitivos y sus Interpretaciones Booleanas

Expresiones

```
[]
{}
a+b
a+b < c**2
True
False
```

```
a=True
b=False
c=False
a and b
a or b
a and b or c
```

Sentencia if

La sentencia *if* de python tiene el siguiente formato:

```
if condición:
  sentencias-condición-verdadera
else:
  sentencias-condición-falsa
```

Si *condición* evalúa verdadero entonces se ejecuta las sentencias *sentencias-condición-verdadera* en caso contrario se ejecutan *sentencias-condición-falsa*

Sentencia if

Python también posee un tipo de sentencia *if* que permite colocar sentencias *if* anidades, es decir una sentencia *if* dentro de otra sentencia *if*.

```
if condición-1:
  sentencias-condición-1-verdadera
elif condición-2:
  sentencias-condición-2-verdadera
elif condición-3:
  sentencias-condición-3-verdadera
else:
  senencias-else
```

Si *condición-1* evalúa verdadero entonces se ejecuta *setencias-condición-1-verdadera*. En caso contrario, si *condición-2* evalúa a verdadera se ejecuta *setencias-condición-2-verdadera*. En caso contrario si *condición-3* evalúa a verdadera se ejecuta *setencias-condición-2-verdadera*. En caso contrario se ejecuta *sentencias-else*

Sentencia if

Ejemplo

```
persona = 'Luke'
if persona == 'Per':
 estado = 'Pythonist'
elif persona == 'Luke':
 estado = 'Jedi knight'
else:
 estado = 'desconocido'
```

Importnte

Pueden haber tantos *elif* como desee el programador.

Iteraciones - for

Python provee una sentencia *for* que tiene la siguiente sintaxis:

```
for i in iterable:
  sentencias-for
else:
  sentencias-else
```

Esta sentencia permite la repetición de un bloque de sentencias indicado por *sentencias-for*. La variable *i* toma cada valor del *iterable* y ejecuta *sentencias-for*. Si el *for* termina normalmente, es decir sin ejecutar una setencia de ruptura (*break* o *return*) se ejecuta *sentencias-else*. En otro caso estas sentencias no se ejecutan.

Sentencia for-Lista

- Repetición de un bloque de sentencias
- Itera a través de una secuencia (lista, tupla, string, diciconarios, iteradores)

Ejemplo

```
s = 0
for i in [0, 1, 2, 3, 4, 5, 6, 7, 8]:
    s = s + 2**i
print ("Suma de Potencias:",s)
```

Sentencia for - Tupla

- Repetición de un bloque de sentencias
- Itera a través de una secuencia (lista, tupla, string, diccionarios, iteradores)

Ejemplo

```
s = 0
for i in (0, 1, 2, 3, 4, 5, 6, 7, 8):
  s = s + 2**i
print ("Suma de Potencias:",s)
```

Sentencia for - String

- Repetición de un bloque de sentencias
- Itera a través de una secuencia (lista, tupla, string, iteradores)

Ejemplo

```
vocal = 0
for i in "Hola Mundo":
  if i.lower() in "aeiou":
    vocal=vocal + 1
print ("La cantidad de vocales es:",vocal)
```

Sentencia for - Diccionarios

- Repetición de un bloque de sentencias
- Itera a través de una secuencia (lista, tupla, string, iteradores)

Ejemplo

```
d={1:10,2:30,3:100,200:500}
valor=0
for j in d.values:
  valor=valor+j
print("Valor:",valor)
```

Sentencia for - Range

Range

- La función **range** es muy útil con la sentencia for.
- **range** crea un iterador que trabaja como una lista.
- Muy eficiente en memoria

Ejemplo

```
s = 0
for i in range(100000):
  if i % 19 == 0:
    s = s + i
print (s)
```

Sentencia for - else

Ejemplo

```
cadena = input("Ingrese una cadena:")
for c in cadena:
 if c  in "0123456789":
  break
else:
 print("La cadena ingresada no contiene números")
print (s)
```

Sentencia for - Sentencias de Ruptura

Ejemplo

```
r = []
cadena=input("Ingrese una cadena:")
for c in cadena:
 if c == ' ': break  #bifurca a la próxima sentencia
                     #después del for
   r.append(c.upper())
   print (''.join(r))
```

Sentencia for - Sentencias de Ruptura

Ejemplo

```
r = []
cadena=input("Ingrese una cadena:")
for c in cadena:
 if c == ' ': continue  #salta el bloque y continua
                        #el loop
    r.append(c)
print (''.join(r))
```

Iteraciones - while

Python provee una sentencia *for* que tiene la siguiente sintaxis:

```
while condición:
  sentencias-while
else:
  sentencias-else
```

Esta sentencia permite la repetición de un bloque de sentencias indicado por *sentencias-while* mientras *condición* sea verdadera. Si el *while* termina normalmente, es decir sin ejecutar una setencia de ruptura (*break* o *return*) se ejecuta *sentencias-else*. En otro caso estas sentencias no se ejecutan.

Sentencia While

Ejemplo

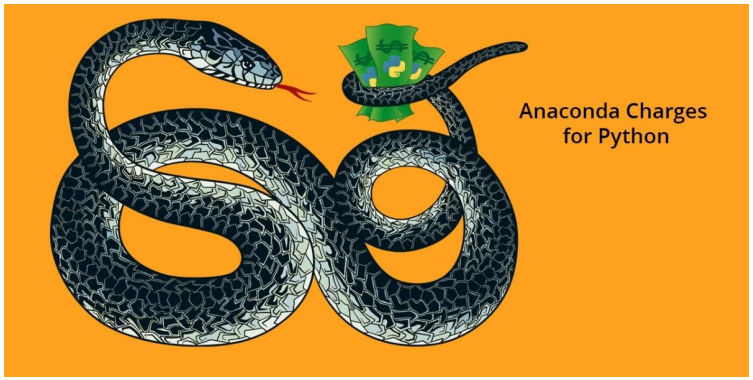
```
r=0
nro=int(input("Ingrese un número positivo:"))
if nro > 0:
  i=0
  while i<nro:
    r=r+2**i
    i=i+1
print("El resultado es:",r)
```

Sentencia While

Ejemplo

```
r = []
n = 0
ult = 20
while n <= ult:     # cualquier expresión
                    # interpretable como boolean
    r.append(str(n))
    n += 3
print (', '.join(r))
```

Sentencias de Repetición



Recordar

- El **else** relacionado con bloque se ejecuta si no se ejecuta un **break**.
- Frecuentemente reemplaza a flags que indican éxito o fracaso.
- Válido en loops **for** y **while**.
- La sentencia **pass** no hace nada.

Sentencias de Repetición

Ejemplo

```
r = [1,2,3,4,5,6]
for i in r:
 if i < 0:
  print 'La entrada contiene un valor negativo!'
  break # sale del loop incluyendo el 'else'
 else:
  pass  # una sentencia que no hace nada
else:   # se ejecuta si el loop termina ok
 print 'input is OK'
```

¿Se anima a hacer este código con un *while*?

Manejo de Errores: try y except

- Los errores de tiempo de ejecución causan serios problemas con la ejecución del programa.
- Los mensajes de error dan el tipo del error.
- **try** y **except** se usan para capturar y manejar errores.
- **try** y **except** modifican el flujo de control del programa.

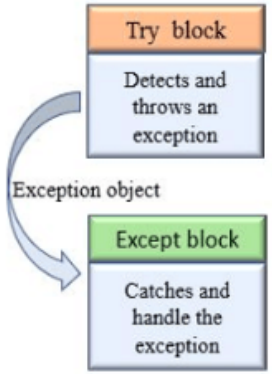
Manejo de Errores: try y except

Ejemplo

```
numeros = []
no_numeros = []
for s in ['12','4.1','1.0e2','e3']
    try:
        n = float(s)
        numeros.append(s)
    except ValueError, msg:
        no_numeros.append(str(msg))
print ('Nros:', numeros)
print ('No Nros:', no_numeros)
```

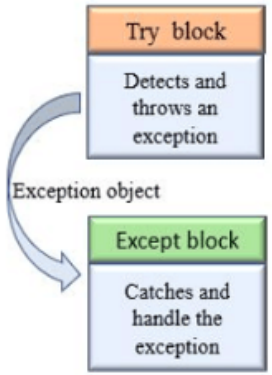
RESULTADO: Nros:['12','4.1','1.0e2'] No Nros: ['invalid literal for float(): e3']

Manejo de Errores: try y except



- Una declaración **try** puede puede tener más de un except para especificar manejadores para más de una excepción.
- Se ejecuta a lo sumo un manejador.
- Solo se manejan excepciones que surgen del **try** correspondiente.
- Un **except** puede nombrar múltiples excepciones usando paréntesis.

Manejo de Errores: try y except




```
... except (RuntimeError, TypeError, NameError):
...     pass
```

Manejo de Errores: try y except

El último **except** puede obviar mencionar que excepción captura para que actúe como comodín.

```
import sys
...
try:
  f = open('miarchivo.txt')
  s = f.readline()
  i = int(s.strip())
except OSError as err:
  print("Error␣OS:␣{0}".format(err))
except ValueError:
  print("No␣pude␣convertir␣el␣dato␣a␣un␣entero.")
except:
  print("Error␣inesperado:", sys.exc_info()[0])
  raise
```

Manejo de Errores: try y except

La declaración **try**...**except** puede tener un else asociado el cual debe ir al final de los **except** y se ejecutará sólo si no se produjo una excepción.

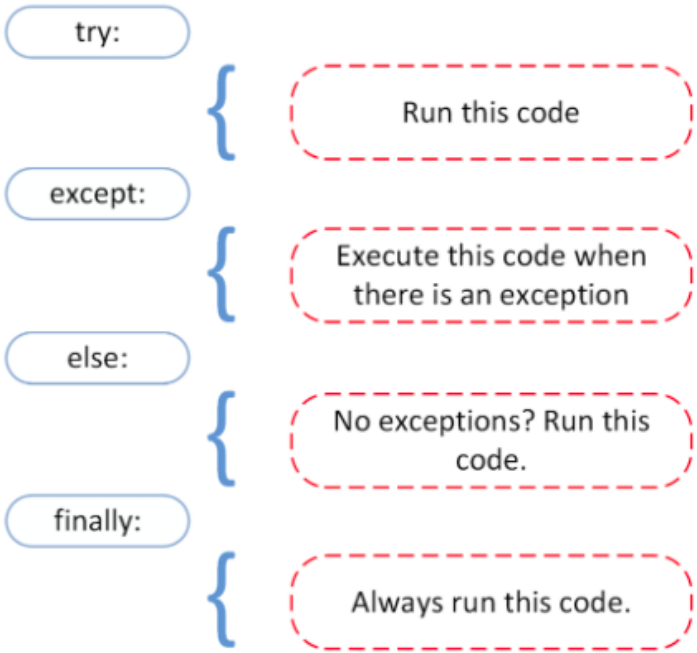
```
for arg in sys.argv[1:]:
try:
  f = open(arg, 'r')
except OSError:
  print('no pude abrir', arg)
else:
  print(arg, 'tiene', len(f.readlines()), 'líneas')
  f.close()
```

Manejo de Errores: try y except

Los manejadores de excepciones no solo manejan las excepciones que ocurren dentro de un bloque **try** sino también de las funciones que se invocan dentro del bloque **try**.

```
>>> def esto_falla():
...     x = 1/0
...
>>> try:
...     esto_falla()
... except ZeroDivisionError as err:
...     print('Manejando error en
                tiempo de ejec:', err)
...
Manejando error en tiempo de ejec.: division by zero
```

Manejo de Errores: try y except



raise permite que programador dispare una excepción. El único argumento de **raise** indica la excepción que se va a disparar.

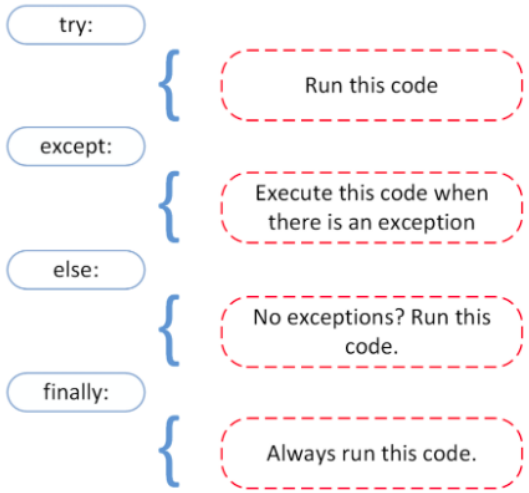
```
>>> raise NameError('Hola')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: Hola
```

Manejo de Errores: try y except

Si se desea determinar si una excepción se disparó pero no se desea manejarla se puede usar **raise** en su forma más sencilla.

```
>>> try:
...     raise NameError('Hola')
... except NameError:
...     print('Excepción!')
...     raise
...
Excepción!
Traceback (most recent call last):
File "<stdin>", line 2, in <module>
NameError: Hola
```

Manejo de Errores: try y except



- La cláusula **finally** siempre se ejecuta antes de que termine el **try** sea que ocurre una excepción o no.
- Cuando ocurre una excepción y la misma no está manejada la misma se vuelve a lanzar después de la ejecución del **finally**.
- **Finally** también se ejecuta cuando se sale del **try except** con un **break**, **continue** o **return**.

Manejo de Errores: try y except

```
try:
 x=int(input("Ingrese un número:"))
 y=int(input("Ingrese un número:"))
 result = x / y
except ZeroDivisionError:
 print("división por cero!")
else:
 print("el resultado es", result)
finally:
 print("ejecutando la cláusula finally")
# si x=2 e y=1
el resultado es 2.0
ejecutando la claúsula finally
# si x2 y=0
división por cero!
ejecutando la cláusula finally
```

¿Cómo dividir grandes líneas?

- Algunas veces las líneas del código fuente necesitan ser divididas.
- Las reglas de indentación prohiben un formato libre

Ejemplo

```
if una_expresión_complicada and
   otra_expresión_complicada:
    print 'Sintaxis ilegal'
```

¿Cómo dividir grandes líneas?

Alternativa 1

Use el caracter \ como último caracter

Ejemplo

```
if una_expresión_complicada and \
    otra_expresión_complicada:
    print ('Sintaxis Válida')
```

¿Cómo dividir grandes líneas?

Alternativa 2

Encerrar las expresiones entre paréntesis.

Ejemplo

```
if (una_expresión_complicada and
    otra_expresión_complicada):
    print('Esta sintaxis es válida')
```