Lenguaje de Programación Python

Sentencias

Dr. Mario Marcelo Berón

Universidad Nacional de San Luis
Departamento de Informática
Área de Programación y Metodologías de Desarrollo de Software

Sentencia if

- La estructura de bloques de Python está determinada por la estructura a bloques.

Ejemplo

```
person = 'Luke'
if person == 'Per':
 status = 'Pythonist'
elif person == 'Luke':
 status = 'Jedi knight'
else:
 status = 'unknown'
print (person, status)
```

Diccionarios como Mejor Alternativa que Sentencias if Anidadas

- Algunas veces es mejor usar diccionarios que sentencias if anidadas.
- Es más compacto, más eficiente.
- Este patrón es muy útil.

Ejemplo

```
status_map = {'Luke': 'Jedi Knight',
              'Per': 'Pythonist'}
person = 'Luke'
print (person, status_map.get(person, 'unknown'))
```

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
- Contenedores vacíos son falsos

Sentencia for

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

Ejemplo

```
s = 0
for i in [0, 1, 2, 3, 4, 5, 6, 7, 8]:
    s = s + i
    if s > 10:
        break                    # Salta al final del loop
print ("i=",i," s=",s)
```

Sentencia for

Ejemplo

```
r = []
for c in 'this is a string with blanks':
  if c == ' ': continue  #salta el bloque y continua
                         #el loop
    r.append(c)
print (''.join(r))
```

Sentencia for

Funciones Primitivas

- 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 While

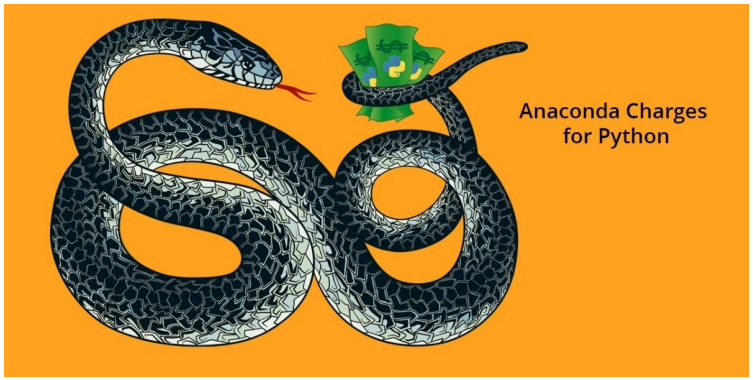
Funciones Primitivas

- Repetición de un bloque de sentencias
- Itera hasta que una condición da como resultado falso o se encuentre una sentencia **break**

Ejemplo

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

Sentencias de Repetición



Funciones Primitivas

- 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'
```

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.

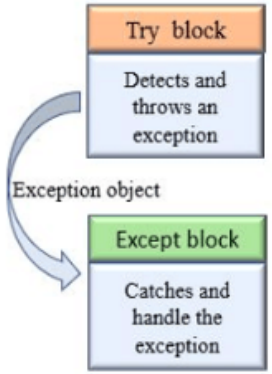
Manejo de Errores: try y except

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)
```

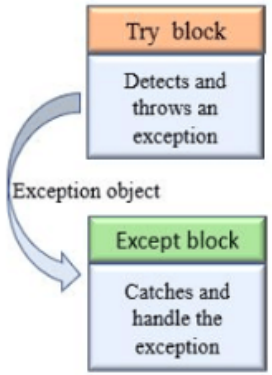
RESULTADO: Nros:['12','4.1','1.0e2'] No Nros: ['invalid literal for float(): e3']

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.

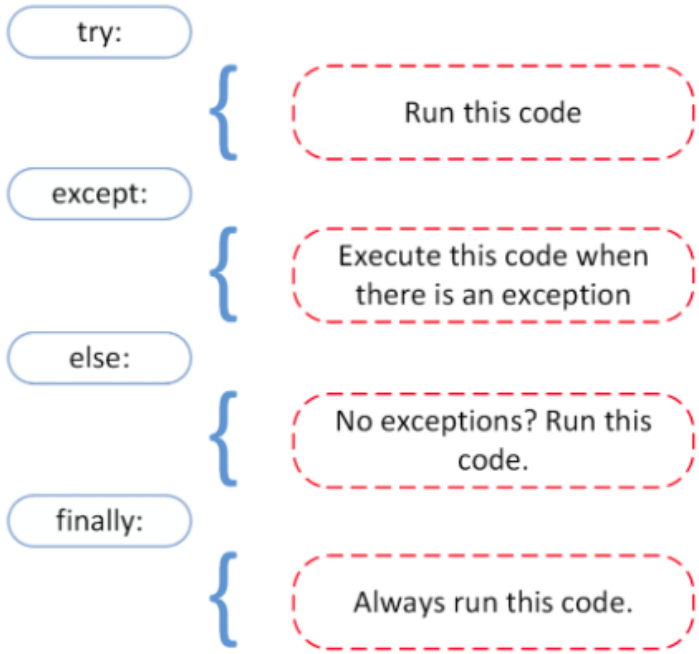
```
for arg in sys.argv[1:]:
try:
  f = open(arg, 'r')
except OSError:
  print('no pude abrir', arg)
else:
  print(arg, 'tiene', len(f.readlines()), 'líneas')
  f.close()
```

Manejo de Errores: try y except

Los manejadores de excepciones no solo manejan las excepciones que ocurren dentro de un bloque **try** sin 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

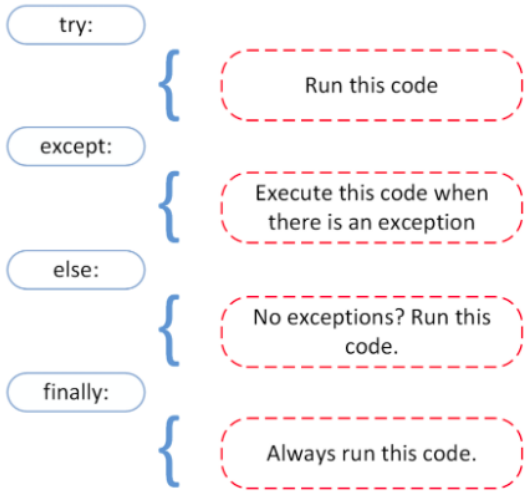
El programador puede definir sus propias excepciones. Para llevar adelante esta tarea se debe definir una subclase de la clase Excepción (o una derivada de esta indirectamente).

```
class Error(Exception):
"""Clase base para excepciones en el módulo."""
 pass
class EntradaError(Error):
"""Excepción lanzada por errores en las entradas.
Atributos:
expresión — expresión de entrada en la que ocurre el
mensaje — explicación del error
"""
 def __init__(self, expresion, mensaje):
  self.expresion = expresion
  self.mensaje = mensaje
```

Manejo de Errores: try y except


```
.....
try:
 raise EntradaError(1,"Este fue un mensaje")
except EntradaError as ee:
 print("Expresion:",ee.expresion,
       " Mensaje:", ee.mensaje)
.....
```

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

```
>>> def dividir(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print("división por cero!")
...     else:
...         print("el resultado es", result)
...     finally:
...         print("ejecutando la cláusula finally")
...
>>> dividir(2, 1)
el resultado es 2.0
ejecutando la claúsula finally
>>> dividir(2, 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_expresi'on_complicada and
   otra_expresi'on_complicada:
    print 'Sintaxis ilegal'
```

¿Cómo dividir grandes líneas?

Alternativa 1

Use el caracter \ como último caracter

Ejemplo

```
if una_expresi'on_complicada and \
    otra_expresi'on_complicada:
    print 'Sintaxis V'alida'
```

¿Cómo dividir grandes líneas?

Alternativa 2

Encerrar las expresiones entre paréntesis.

Ejemplo

```
if (una_expresi'o_complicada and
    otra_expresi'on_complicada):
    print 'Esta sintaxis es v'alida'
```