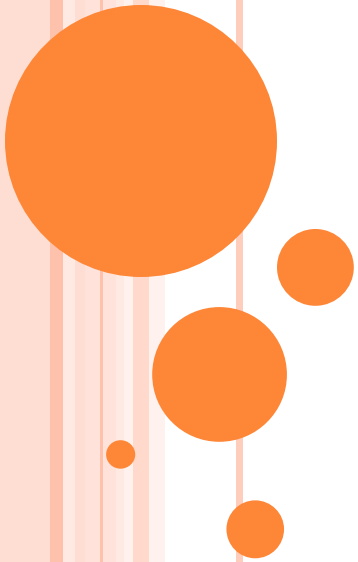


Excepciones

CYPLP

EXCEPCIONES

¿Qué es una excepción?

Condición inesperada o inusual, que surge durante la ejecución del programa y no puede ser manejada en el contexto local.

EXCEPCIONES

¿Qué debemos tener en cuenta sobre un lenguaje que provee manejo de excepciones?:

- ¿**Cómo** se **maneja** una excepción y cuál es su **ámbito**?
- ¿**Cómo** se **alcanza** una excepción?
- ¿Cómo **especificar** la unidades (manejadores de excepciones) que se han de ejecutar cuando se alcanza las excepciones?
- ¿A dónde se **cede el control** cuando se termina de atender las excepciones?
- ¿Cómo se **propagan** las excepciones?
- ¿Hay excepciones **predefinidas**?

EXCEPCIONES

- Dos modelos de ejecución
  - Continuación
    - PL/1
  - Terminación
    - ADA
    - CLU
    - C++
    - Java
    - Phyton
    - PHP

Algunos lenguajes que incorporaron el manejo de excepciones- PL/I

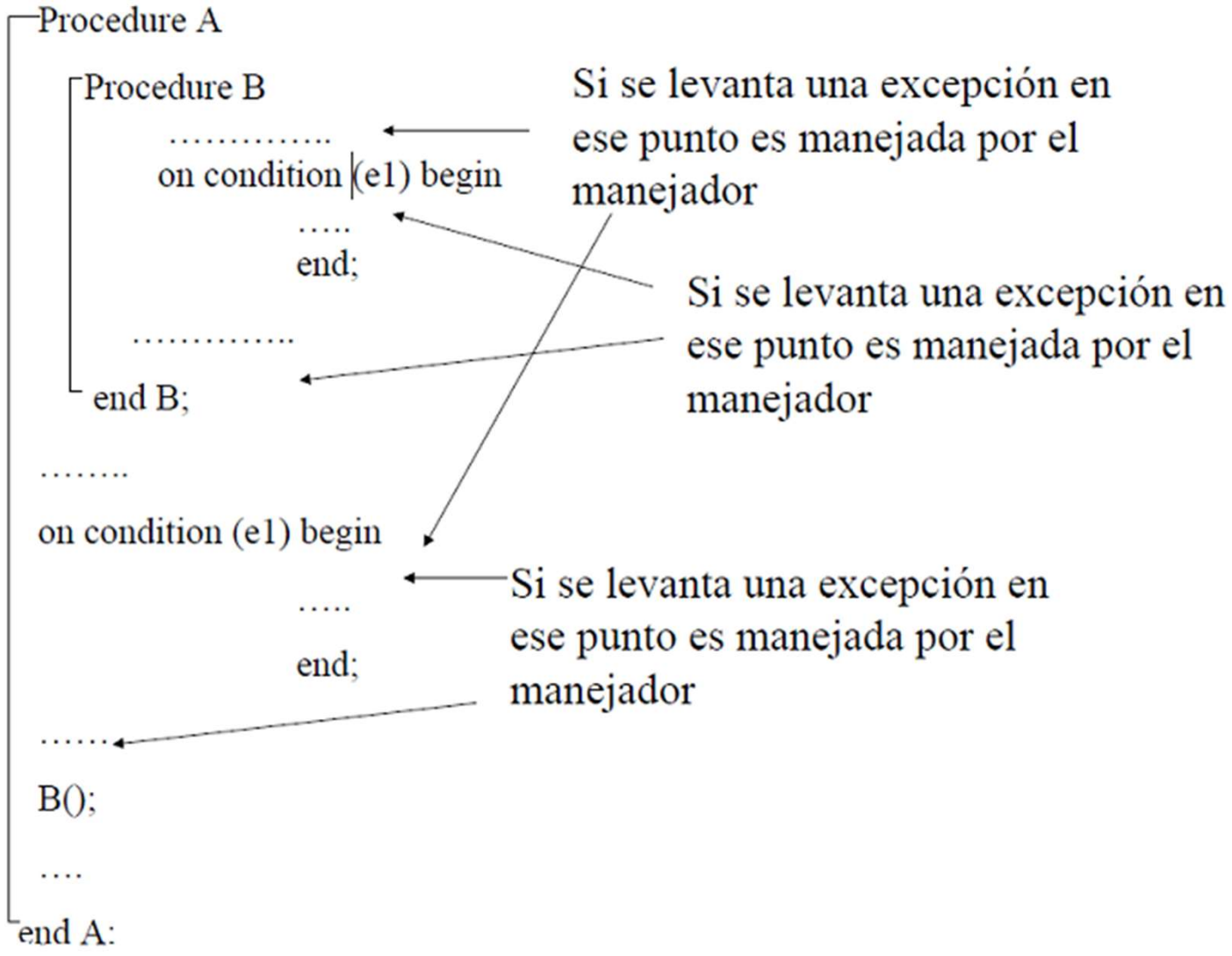
- Fue el primer lenguaje. que incorporó el manejo de excepciones.
- Utiliza el criterio de **Reasunción. Cada vez que se produce la excepción,** la maneja el manejador y devuelve el control a la sentencia **siguiente** de dónde se levantó.
- La excepciones son llamadas CONDITIONS
- Los manejadores se declaran con la sentencia ON:

  ON CONDITION(Nombre-excepción) Manejador
- El manejador puede ser una instrucción o un bloque
- Las excepciones se alcanzan explícitamente con la palabra clave **Signal condition(Nombre-excepción)**

EXCEPCIONES – PL/I

- Este lenguaje tiene una serie de excepciones ya **predefinidas con su** manejador asociado. Son las **Built-in exceptions.**
- **Por ej. zerodivide, se levanta cuando hay una división por cero.**
- Los manejadores se ligan **dinámicamente con las excepciones. Una** excepción siempre estará ligada con el último manejador definido.
- El alcance de un manejador termina cuando finaliza la ejecución de la unidad donde fue declarado

EXCEPCIONES – PLI EJEMPLO



EXCEPCIONES – ADA

- Criterio de **Terminación.** Cada vez que se produce una excepción, se **termina el bloque dónde se** levantó y se ejecuta el manejador asociado.
- Se definen en la zona de definición de las variables y tienen el mismo alcance
- Se alcanzan explícitamente con la palabra clave **raise**
- Los manejadores pueden encontrarse en el final de cuatro diferentes unidades de programa: **Bloque, Procedimiento, Paquete o Tarea.**

```
Procedure prueba;
      e1, e3: Exception
Begin
    .......
    Exception
    when e1 → Manejador1;
    when others →Manejadorn;
End;
```

EXCEPCIONES – ADA

- Tiene cuatro excepciones predefinidas : **Constraint_error, Program_error, Storage_error, Numeric_Error, Name_Error, Tasking_error.**
- **Propagación**, al producirse una excepción:
  - Se termina la unidad, bloque, paquete o tarea dónde se alcanza la unidad.
  - Si el manejador **se encuentra en ese ámbito,** se ejecuta**.**
  - Si el manejador **no se encuentra en ese lugar la excepción se propaga dinámicamente. Esto significa que se vuelve a levantar en** otro ámbito.
  - Siempre tener en cuenta el alcance, puede convertirse en anónima!
- Una excepción se puede levantar nuevamente colocando solo la palabra **raise.**

EXCEPCIONES – ADA

```
procedure A is
  x,y,a : integer;
  e1, e:exception;
  procedure B (x : integer; y: integer) is
     m : integer;
     e:exception;
        begin
        ....
        Exception
                when e * x:=x+1: raise;
                when e1 * null;
             end B;
 begin
  ......
Exception
                when e * x:=x+1;
                when others * x:=x+2;
 end A;
```

Si se levanta la excepción **e …**

Es manejada primero por el manejador e de B y luego por el manejador others de A

Pueden programarse Manejadores que no realicen nada. Usando la sentencia "null". Ejemplo..

EXCEPCIONES – ADA

Tener en cuenta :

- Es determinístico en la asociación de la excepción con el manejador.
- Si se deseara continuar ejecutando las instrucciones de un bloque donde se lanza una excepción, es preciso “crear un bloque más interno”, como se muestra en el ejemplo:

```
Procedure Bloque () is
....
begin
…
 Declare
  …..
  begin -- del bloque interno
     instrucciones que pueden
   fallar
        exception
              manejadores
end; -- del bloque interno
….
Instrucciones que es preciso
   ejecutar, aunque se haya
levantado la excepción en el
   bloque interno
….
end;
```

Se pueden definir nuevas excepciones

EXCEPCIONES – CLU

- Utiliza el criterio de **Terminación.**
- Solamente se pueden **ser alcanzadas** por los **procedimientos**.
- Están asociadas a **sentencias**.
- La excepciones que un procedimiento puede alcanzar se declaran en su encabezado.
- Se alcanzan explícitamente con la palabra clave **signal**
- Los **manejadores** se **colocan** al lado de una **sentencia** simple o compleja. Forma de definirlos:

```
<sentencia> except
when Nombre-Excepción: Manejador1;
when Nombre-Excepción : Manejador2;
.....
when others: Manejadorn;
end;
```

EXCEPCIONES – CLU

- Posee excepciones **predefinidas** con su manejador asociado. Por ejemplo failure
- Se pueden pasar parámetros a los manejadores.
- Una excepción se puede volver a levantar una sola vez utilizando **resignal**
- Una excepción se puede levantar en cualquier lugar del código

```
Procedure ejemplo () signals e1, e2
Begin
……
if ( .. ) then A(); except
                    when e1: Manejador1;
                    when e2 : Manejador2;
                    when others : Manejador3;
          end;
….
End;
```

EXCEPCIONES – CLU

Propagación, al producirse una excepción:

- Se termina el procedimiento donde se levantó la excepción y devuelve el control al llamante inmediato donde **se debe encontrar el** manejador.
- Si el manejador **se encuentra en ese ámbito, se ejecuta y luego se** pasa el control a la sentencia siguiente a la que está ligado dicho manejador.
- Si el manejador **no se encuentra en ese lugar la excepción se propaga estáticamente** en las sentencias asociadas**.** Esto significa que el proceso se repite para las sentencias incluidas estáticamente.
- En caso de no encontrar ningún manejador en el procedimiento que hizo la llamada se levanta una excepción **failure** y devuelve el control, terminando todo el programa

EXCEPCIONES – CLU

```
Procedure Main
  .....
Procedure UNO() signals error1;
        x:integer
        Begin
        x:=2; ......
        While y < x Do
          If y=0 Then  signal error1;
          end if;
          exception when error1: <sent>-;
                          resignal;
           end;  // manejador4
           Dos();
        Wend; exception
                  when error1: <sent> .End;
    End; //UNO
```

UNO termina

Ejecuta este manejador

```
Procedure Dos() signals error1;
    m:integer;
    Begin
      ...
      if m=0 then signal error1;
      End;

Begin //MAIN
  x:=1; y:=0;
Uno();
 exception  when  error1:  x:=x+1;
 y:=y+1; end;
...
Dos():
exception  when  error1:  resignal;
 end;
...  End; //MAIN
```

EXCEPCIONES – C++

- Utiliza el criterio de **Terminación.**
- Las excepciones pueden alcanzarse explícitamente a través de la sentencia **throw**
- Posee excepciones predefinidas
- Los manejadores van **asociados a bloques** .
- Los bloques que pueden llegar a levantar excepciones van precedidos por la palabra clave **Try** y al finalizar el bloque se detallan los manejadores utilizando la palabra clave **Catch(NombreDeLaExcepción)**
- Al levantarse una excepción dentro del bloque **Try el control se transfiere** al manejador correspondiente.
- Al finalizar la ejecución del manejador la ejecución continúa como si la unidad que provocó la excepción fue ejecutada normalmente.
- Permite pasar **parámetros al levantar la excepción.**

  Ejemplo: **Throw (Ayuda msg);**

  Se está levantando la excepción Ayuda y se le pasa el parámetro msg.

EXCEPCIONES – C++

Ejemplo:

```
…
Try
{
….. /* Sentencias que pueden provocar una excepción*/
}
catch(NombreExcepción1)
{
….. /* Sentencias Manejador 1*/
}
….
catch(NombreExcepciónN)
{
….. /* Sentencias Manejador N*/
}
…
```

Las excepciones se pueden levantar nuevamente colocando **Throw.**

EXCEPCIONES – C++

- Las rutinas en su interface pueden listar las excepciones que ellas pueden alcanzar.

void rutina ( ) throw (Ayuda, Zerodivide);

¿Qué sucede si la rutina …?

- **alcanzó otra excepción que no está contemplada en el listado de la Interface?**

  En este caso **NO se propaga la excepción y una función especial se** ejecuta automáticamente: **unexpected(), que generalmente causa abort(),** que provoca el final del programa. Unexpected puede ser redefinida por el programador.

- **Colocó en su interface el listado de posibles excepciones a alcanzar**

  En este caso **Si se propaga la excepción. Si una excepción es** repetidamente propagada y no machea con ningún manejador, entonces una función **terminate()** es ejecutada automáticamente.

- **Colocó en su interface una lista vacía (throw())?**

  Significa que **NINGUNA excepción será propagada.**

EXCEPCIONES - JAVA

- Al igual que C++ las excepciones son objetos que pueden ser alcanzados y manejados por manejadores adicionados al **bloque** donde se produjo la excepción.
- Cada excepción está representada por una instancia de la clase **Throwable** o de una de sus subclases (Error y Exception)
- La gestión de excepciones se lleva a cabo mediante cinco palabras clave: **try, catch, throw, throws, finally**.
- Se debe especificar mediante la cláusula **throws** cualquier excepción que se envía desde un método.
- Se debe poner cualquier código que el programador desee que se ejecute siempre, en el método **finally**.

Fases del tratamiento de Excepciones

- **Detectar e informar del error:**
  - Lanzamiento de Excepciones → throw
  - Un método detecta una condición anormal que le impide continuar con su ejecución y finaliza "lanzando" un objeto Excepción.
- **Recoger el error y tratarlo:**
  - Captura de Excepciones → bloque try-catch
  - Un método recibe un objeto Excepción que le indica que otro método no ha terminado correctamente su ejecución y decide actuar en función del tipo de error.

EXCEPCIONES - JAVA

Orden correcto

```
try{
 ...
}
Catch (ArthmeticException e)
        {   trataArithmeticException(e);}
catch (Exception e2)
        {   trataTodasLasExcepciones(e2); }
Finally
        {   bloque final; }
```

Puede estar o no
Si está, la ejecución de su código se realiza cuando se termina la ejecución del bloque Try, se haya o no levantado una excepción. Salvo que el bloque Try haya levantado una excepción que no macheo con ningún manejador.

Propagación, bloques Try anidados

```
// demuestra cómo lanzar una excepción cuando ocurre una división entre cero
   public int cociente( int numerador, int denominador )
      throws ArithmeticException
   {
      return numerador / denominador;
   }
```

EXCEPCIONES - PHYTON

- Se manejan a través de bloques try except

```
>>> while True:
...     try:
...         x = int(input("Por favor ingrese un número: "))
...         break
...     except ValueError:
...         print("Oops! No era válido. Intente nuevamente...")
```

La declaración **try** funciona de la siguiente manera:

- Primero, se ejecuta el *bloque try* (el código entre las declaración **try** y **except**).
- Si no ocurre ninguna excepción, el *bloque except* se saltea y termina la ejecución de la declaración **try**.
- Si ocurre una excepción durante la ejecución del *bloque try*, el resto del bloque se saltea. Luego, si su tipo coincide con la excepción nombrada luego de la palabra reservada **except**, se ejecuta el *bloque except*, y la ejecución continúa luego de la declaración **try**.
- Si ocurre una excepción que no coincide con la excepción nombrada en el **except**, esta se pasa a declaraciones **try** de más afuera; si no se encuentra nada que la maneje, es una *excepción no manejada*, y la ejecución se frena con un mensaje como los mostrados arriba.

EXCEPCIONES - PHYTON

- Presenta la siguiente estructura para manejo de excepciones:

```
try:
        sentencia 1
        ....
        sentencia n
except nombre de la  excep1 as var:
        sentencias
        ..
except nombre de la excepción n :
        sentencias
else:
        sentencias
finally:
        sentencias
```

Un conjunto de excepciones pueden ser manejadas por un mismo manejador. En ese caso se puede colocar:

```
except (exp1,exp2,..):
```

Puede aparecer un except **SIN nombre** de excepción, pero **SOLO** al final. Actúa como comodín

```
except:
```

El código colocado en la cláusula **else** se ejecuta **solo si** no se levante una excepción

El código colocado en la cláusula **finally** se ejecuta **siempre**

Opcional

EXCEPCIONES - PHYTON

¿Qué sucede cuando una excepción no encuentra un manejador en su bloque "try except"?

- **Busca estáticamente**

  Analiza si ese try está contenido dentro de otro y si ese otro tiene un manejador para esa excepción. Sino...

- **Busca dinámicamente**

  Analiza quién lo llamó y busca allí

- **Si no se encuentra un manejador, se corta el proceso y** larga el mensaje standard de error

- **Levanta excepciones explícitamente con "raise"**

EXCEPCIONES - PHYTON

- **Ej. con else:**

Cláusula **opcional**, que se ejecuta **SOLO** si **NO** se levanta excepción en el bloque try except

PruebaConElse.py - E:\Python\Clases\Pruebas\Excepciones\PruebaConElse.py

File Edit Format Run Options Windows Help

```
dic={1:'juan',4:'pedro',5:'helena'}
y=8
print y
try:
    for x in range(1,6):
        print dic[x]
except (KeyError):
    dic[x]='nuevo'
else:
    print 'Se ejecutó el bloque try except en forma correcta'
print dic
```

Ln: 12 Col: 0

Se levanta la excepción..

```
>>>
8
juan
{1: 'juan', 2: 'nuevo', 4: 'pedro', 5: 'helena'}
```

→ **NO se ejecuta** el bloque **else**

PruebaConElseOtro.py - E:/Python/Clases/Pruebas/Excepciones/PruebaConEl...

File Edit Format Run Options Windows Help

```
dic={1:'juan',4:'pedro',5:'helena'}
y=8
print y
try:
    for x in (1,4,5):
        print dic[x]
except (KeyError):
    dic[x]='nuevo'
else:
    print 'Se ejecutó el bloque try except en forma correcta'
print dic
```

Ln: 5 Col: 19

NO se levanta la excepción..

```
>>>
8
juan
pedro
helena
Se ejecutó el bloque try except en forma correcta
{1: 'juan', 4: 'pedro', 5: 'helena'}
```

→ **SI se ejecuta** el bloque **else**

Excepciones en PHP

- Modelo de Terminación
- Una excepción puede ser lanzada (thrown), y atrapada ("catched")
- El código esta dentro de un bloque try,
- Cada bloque try debe tener al menos un bloque catch correspondiente.
- Las excepciones pueden ser lanzadas (o relanzadas) dentro de un bloque catch.
- Se puede utilizar un bloque finally después de los bloques catch

Excepciones en PHP

- El objeto lanzado debe ser una instancia de la clase Exception o de una subclase de Exception. Intentar lanzar un objeto que no lo es resultará en un Error Fatal de PHP.
- Cuando una excepción es lanzada, el código siguiente a la declaración no será ejecutado, y PHP intentará encontrar el primer bloque catch coincidente. Si una excepción no es capturada, se emitirá un Error Fatal de PHP con un mensaje "Uncaught Exception ..." ("Excepción No Capturada"), a menos que se haya definido un gestor con set_exception_handler().

Excepciones en PHP

```php
<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('División por cero.');
    }
    return 1/$x;
}
try {
    echo inverse(5) . "\n";
} catch (Exception $e) {
    echo 'Excepción capturada: ',  $e->getMessage(), "\n";
} finally {
    echo "Primer finally.\n";
}
try {
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Excepción capturada: ',  $e->getMessage(), "\n";
} finally {
    echo "Segundo finally.\n";
}
// Continuar ejecución
echo 'Hola Mundo\n';
?>
```

El resultado del ejemplo sería:

```
0.2
Primer finally.
Excepción capturada: División por cero.
Segundo finally.
Hola Mundo
```

Lenguajes que NO proveen manejo de Excepciones - Simular!!

```
procedure A is
   x,y,a : integer;
   e1, e: exception;
  procedure B (x : integer; y: integer) is
     m : integer;
     e: exception;
   begin  ....
     if (m <=0) then  if (m= 0) then raise e;
                                else raise e1;
          ...
       Exception
         when e * Manejador 3;
    end B;
  begin
    ..... B(x,y);
  Exception
          when e * Manejador 1;
          when others * Manejador 2;
end A;
```

```
procedure Manejador 1 ();
          begin
          ----
          end;
procedure Manejador 2 ();
          begin
          ----
          end;
```

```
procedure Manejador 3 ();
                    begin
                    ----
                    end;
```

```
if (m <=0) then  if (m= 0) then Manejador 3();
                           else Manejador 2();
          else .....
```

Programa escrito en ADA

DUDAS?

- Cual modelo les parece mas seguro?
- Les parece útil el manejo de excepciones? Lo creen necesario?
- El modelo de terminación tiene [illegible] comportamientos