

Repaso de Pascal

Introducción a Python

Lenguajes de Programación 2019

PASCAL

Pascal - introducción

- Pascal es un Lenguaje basado en el **paradigma imperativo clasico**
 - Estructura la programación mediante
 - Funciones
 - Procedimientos
 - Estructuras de Control
 - Estructuras de Datos
- Posee chequeo estático de tipos



Niklaus Wirth
Creador de Pascal

Pascal

- Pascal **divide** sus programas en:
- **Declaración:**
 - Una sección en donde se declaran todas las **variables, tipos, constantes, funciones y procedimientos** que se pueden utilizar en el programa
- **Ejecución:**
 - La parte ejecutiva del programa es un conjunto estructurado de sentencias donde se pueden utilizar los elementos declarados en la sección de **declaración**

Pascal – Declaraciones: Tipos

- Declaración de Tipos:

type

miEntero = Integer;

miEnumerado = (ingeniero, futbolista);

miSubrango = 1..99;

miArreglo = array [1..99] of integer;

miRegistro = **record**

 DNI: Integer

 Edad: Integer

case Profesion: miEnumerado **of**

 ingeniero: (Matricula: miEntero)

 futbolista: (Club: string)

end;

Pascal – Declaraciones: Variables

- Las **variables** en Pascal son **explícitamente declaradas**

Var

```
cliente, amigo: miRegistro;  
contador: miEntero;
```

Pascal – Declaraciones: Procedimientos y Funciones

- Para especificar **subprogramas** Pascal permite declarar:

Procedimientos:

```
procedure miProc (a: Integer, var b: Integer);
```

```
type  tipoInterno = Integer
```

```
var   varLocal: tipoInterno
```

```
Begin
```

```
....
```

```
End;
```

Pasaje por
Valor

Pasaje por
Referencia

Declaraciones
Internas

Bloque
Código de
miProc


En Pascal los bloques
se delimitan con
Begin - End

Pascal – Declaraciones: Procedimientos y Funciones

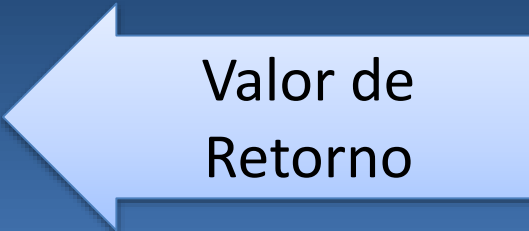
- Para especificar subprogramas Pascal permite declarar:

Funciones:

```
function miFunc (a: Integer, var b: Integer): Integer;  
    type    tipoInterno = Integer  
    var     varLocal: tipoInterno  
    Begin  
        ...  
        miFunc := 12378;  
    End;
```



Tipo de
Retorno



Valor de
Retorno

Pascal - Sentencias

- Asignación

V1:= V2 + 33;

- Secuencia

V1:= V2 + 33;

V6:= 'pruebas';

- Condicional

If (v3 and v2 > v1 or v6 = v7) Then

Begin

....

End;

Pascal - Sentencias

- Repetición

While ($x > y$) **do**

Begin

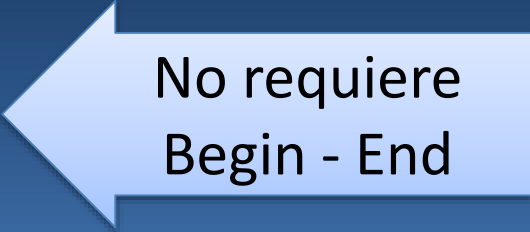
....

End;

Repeat

... Secuencia de Sentencias...

Until ($x < y$);



No requiere
Begin - End

Pascal - Sentencias

- Repetición

For contador := 1 to 20 do

Begin

...

End;

Pascal - Sentencias

- Las **llamadas a procedimiento** son sentencias simples

```
V1:= V2 + 33;  
miProc(V1, V6);
```

- Las **llamadas a función** se realizan en el contexto de una expresión:

```
V1:= V2 + miFunc(2,N);
```

PYTHON

Python

- Python es un lenguaje de Scripting que surge en los años 90
 - Es normalmente, utilizado para administrar sistemas, programación de CGI, lenguaje de scripting interno, programación web, computación científica, procesamiento de lenguaje natural
 - Multiparadigma: Imperativo, Objetos y Funcional



Guido Van Rossum
Creador de Python

Python

- Muy legible y escribible
 - Sintaxis minimalista y elegante
 - Considerablemente Ortogonal
 - Muy Flexible
- Posee tipado dinámico
 - Aun así, es confiable ya que detecta todos los errores de tipo
- Interpretado



Guido Van Rossum
Creador de Python

Python - Interprete

<http://www.python.org>

- Modo Interactivo: python.exe / IDLE

```
>>> 5+4
```

```
9
```

```
>>> X = 'prueba'
```

```
>>> X
```

```
'prueba'
```


Python - Variables

- En Python las **variables** son **implícitamente declaradas** cuando aparecen del lado izquierdo de una asignación

```
>>> X = 50
```

```
>>> X = 'messi hace goles'
```

```
>>> X = X + Y (Error: Y no esta declarado)
```

Python - Variables

- La **ligadura** entre una **variable y su tipo** se produce **dinámicamente**

>>> X = 50

Entero

>>> X = 'messi hace goles'

String

>>> X = [1, 5, 'la verdad', [5,4], 10]

Lista

Python – Tipos Predefinidos

- **Numéricos:** Enteros, Enteros largos, Floats, y Complejos

```
>>> X = 5
```

```
>>> X = 4L
```

```
>>> X = 0.566
```

```
>>> X = complex(0.4, 8)
```

Python – Tipos Predefinidos

- **Strings:** Cadenas de caracteres denotadas por `' '` o `"""`

```
>>> X = 'python'
```

```
>>> X + ' lindo lenguaje'  
'python lindo lenguaje'
```

Concatenación

```
>>> X[1]  
'y'
```

```
>>> X[1:4]
```

Indización y Slices

```
'yth'
```

```
>>> X[1:4] = 'ath' (Error de Tipos)
```

Los Strings
son Inmutables

Python - Tipos Predefinidos

Las Listas en Python pueden verse como arreglos dinámicos

- **Listas:** Colecciones ordenadas mutables de elementos heterogéneos delimitadas por `[]`

```
>>> X = [1, 'codo', 5, 0]
```

```
>>> X + ['rojo', 'a']
```

```
[1, 'codo', 5, 0, 'rojo', 'a']
```

Concatenación

```
>>> X[1:3]
```

```
['codo', 5]
```

Indización y Slices

```
>>> X[1:3] = ['a', 1]
```

```
>>> X[0] = 'cambio'
```

```
>>> X
```

```
['cambio', 'a', 1, 0]
```

Los Listas son Mutables

Python - Tipos Predefinidos

- **Tuplas:** Colecciones ordenadas inmutables de elementos, delimitados por ()
- `>>> X = (1, 'codo', 3)`
- Se **manipulan** de manera **similar a los Strings y Listas**, permitiendo Concatenación, Producto, Indización y Slices
- Al igual que los Strings **son Inmutables**

Python – Tipos Predefinidos

- **Diccionarios:** Son arreglos asociativos (mapeos) de elementos, denotados usando { }

```
>>> oficinas = { 'gotti': 17, 'laura': 4 }
```

```
>>> oficinas['gotti']
```

```
17
```

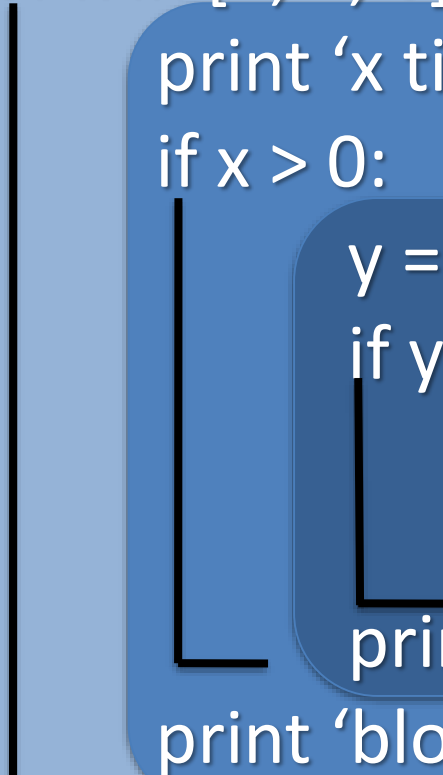
- A diferencia de las listas, strings y tuplas estan **indizados por llaves**, que pueden ser de cualquier **tipo inmutable**.

Python – Bloques y Sentencias

- En Python las sentencias se estructuran mediante **bloques anidados**
- A diferencia de otros lenguajes Python **delimita** sintácticamente **los bloques mediante indentación** a través de
 - Tabs, o
 - Espacios (4)

Python – Bloques y Sentencias

```
for x in [2, 1, 0]:  
    print 'x tiene', x  
    if x > 0:  
        y = 2  
        if y==x:  
            print 'bloque2'  
            print 'mas bloque2'  
        print 'bloque1'  
    print 'bloque0'
```



The diagram illustrates the nesting of code blocks in Python. It uses L-shaped brackets to show the scope of each statement. The outermost block is the 'for' loop. Inside it is an 'if' statement block. Inside the 'if' statement is another 'if' statement block. Inside the innermost 'if' statement are two 'print' statements. The blocks are represented by nested rounded rectangles, with the innermost block being the darkest blue and the outermost being the lightest blue. The L-shaped brackets are black and connect the start of each line of code to its corresponding closing bracket, showing the hierarchy of the code blocks.

Python - Asignacion

- La **asignación** se realiza mediante =

```
>>> X = 12  
>>> X = 'viva la patria'
```
- Además se permiten **multi-asiganciones**

```
>>> U, V, W = 1, 2, 3  
>>> X, Y = 50, X+10
```

Python - Condicional

```
if x < 0:  
    x = 0  
    print 'Ahora es Cero'  
elif x == 0:  
    print 'Cero'  
elif x == 1:  
    print 'Es un 1'  
else:  
    print 'Es mas grande'
```

```
if <condición>:  
    <bloque if>  
elif <condición2>:  
    < bloque elif>  
else:  
    < bloque else>
```

Python - While

```
while b < 10:  
    print b  
    a, b = b, a+b
```

```
while <condición>:  
    < bloque while>
```

Python - For

```
for x in [1, 3, 5, 7]:  
    print x
```

```
>>> range(1,9)  
[1,2,3,4,5,6,7,8,9]
```

La función range
construye una lista
en base a los limites

```
for x in range(1,9):  
    print x
```

```
for <destino> in <objeto>:  
    <bloque for>
```

Python - Funciones

```
def fib(n):  
    a, b = 0, 1  
    while a < n:  
        print a  
        a, b = b, a+b
```

```
def <nombre> (<parametros>):  
    <bloque funcion>
```

Python - Funciones

```
def fib(n):  
    a, b = 0, 1  
    while a < n:  
        print a  
        a, b = b, a+b  
    return a
```

La definición de una función es una sentencia mas!

Python – Alcance, Visibilidad y Ambientes de Referenciamiento

- El **alcance en Python es estático**
- Esta determinado por el **anidamiento de definiciones de funciones**

```
a = 3
```

```
def sub1():
```

```
    a = 5
```

```
    b = 7
```

```
    c = 15
```

```
    x = a + b + c
```

```
    def sub2():
```

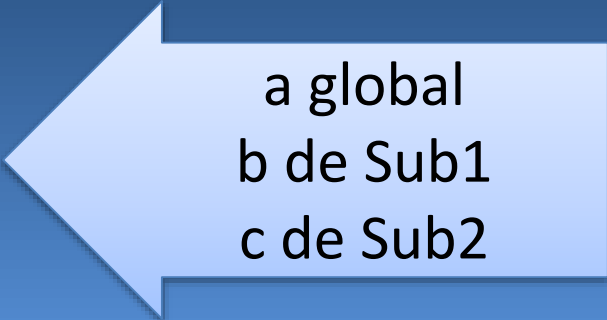
```
        global a
```

```
        c = 12
```

```
        z = a + c
```



a, b, c de Sub1



a global
b de Sub1
c de Sub2

Python - Pasaje de Parámetros

- El pasaje de parámetros **es por Valor**
- Sin embargo, las **variables con tipos estructurados son referencias**

```
def pasaje(x, y):
```

```
    x = 2
```

```
    y[0] = 2
```

```
a = 1
```

```
b = [1,2,3]
```

```
pasaje(a, b)
```

```
print 'a=', a, 'b=', b
```

```
a= 1  
b= [2,2,3]
```

Python – Pasaje de Parámetros

- Los parámetros pueden tener valor por **Default**



```
def defaultP(x, y=10):  
    return x + y
```

defaultP(2,3)

Retorna 5

defaultP(2)

Retorna 12

defaultP()

Error

No puede haber un parámetro **sin default** a **derecha** de uno con **default!!!**

Python – Pasaje de Parámetros

- Los parámetros se pueden usar pasando por **palabra clave**

```
def concat(x, y, z):  
    return x + y + z
```

concat('a', 'b', 'c') ← Retorna 'abc'

concat('a', z='b', y='c') ← Retorna 'acb'

concat('a', w='c', z='b') ← Error

Python – Funciones

- Las **funciones son objetos** para python, por lo tanto **pueden ser asignadas**

```
def mostrar(cartel):  
    print cartel
```

```
prueba = mostrar  
prueba('asignada')
```

asignada

Python – Funciones

- También pueden ser **pasadas por parámetro**

```
def aplicarFuncion(func, arg):  
    func(arg)
```

```
def mostrar(cartel):  
    print cartel
```

```
aplicarFuncion(mostrar, 'esto es una prueba')
```

esto es una prueba

Referencias

- **The Python Tutorial**

- <https://docs.python.org/2/tutorial/index.html>