CAPACITACIÓN PYTHON

NELSON R. SALINAS

IDEAM - SMBYC

OBJETIVO

- Presentar una introducción a Python bajo la cual el conocimiento computacional no sea un prerrequisito.
- Algunos temas serán sobresimplificados, otros eludidos.

¿POR QUÉ PYTHON?

- Lenguaje de programación multipropósito.
- Sintaxis fácil de aprender.
- Generación de alertas especialmente útiles para el programador novato.
- Gran número de librerías adicionales que extienden o máximizan la utilidad.

INTRODUCCIÓN

1. Bases
2. Tipos de datos
3. Operaciones básicas
4. Operadores, expresiones
5. Variables

BASES

Editor de texto. ¿Cómo debe ser?

- Codificación (ASCII, UTF-8).
- Caracter de fin de línea.
- Enfasis de sintático específico a cada lenguaje.
- JEdit, Geany, Atom.
- *Jamás* Microsoft Word o similares.

PYTHON

- Creado en 1991 por Guido van Rosum.
- Lenguaje de programación de alto nivel.
- Amplia diversidad de aplicación (librerías).
- Intrepretado, no compilado.
- Versiones populares: 2.7 y 3.6.

PYTHON

```
def chaveI_forest(precipitation):
    """Estimates Chave et al. 2005 forest type."""

    out = None

            if precipitation <= 1500:
                    out = 'dry'
            elif precipitation <= 3500:
                    out = 'moist'
            else: # greater than 3500
                    out = 'wet'

            return out
```

PYTHON

Consola interactiva

```
Python 2.7.14 | (default, Oct 16 2017, 17:29:19)
[GCC 7.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more info
>>> |
```

TIPOS DE DATOS

1. Escalares (elementos indivisibles).
    - int: `1`, `0`.
    - float: `1.5`, `3.14259`, `6.022e-23`.
    - bool: `True`, `False`.
    - None: `None`.

TIPOS DE DATOS

1. No escalares (elementos divisibles).
    - list: `[1, 2, 3, 4]`
    - tuple: `(1, 2, 3, 4)`
    - dict: `{1: 125, 2: 345, 3: 1023}`
    - str: `"Hola",'Ciao'`
    - unicode: `u"Hola",u'Ciao'`

TIPOS DE DATOS

1. No escalares (elementos divisibles).

list, tuple, dict => subunidades de diferentes tipos de datos.

```
[1, 's', u's', 4.9]
{'s': 125, 2.0: 345, 3: u's'}
```

FUNCIONES BÁSICAS

len:

tamaño del objecto.

```
len([1,2,3])
# 3
len([])
    # 0
len({1: 125, 2: 345, 3: 1023}) # 3
len('Hola')
# 4
```

FUNCIONES BÁSICAS

Índices y slicing:

Acceso a subcomponentes de no scalares (list, dict, etc.).

```
mylist = [1, 5, 7]
mylist[0] #  1
mylist[2] #  7
mylist[-1] #  7
```

FUNCIONES BÁSICAS

Índices y slicing:

Acceso a subcomponentes de no scalares (list, dict, etc.).

```
mylist = [1, 3, 5, 7, 9]
mylist[:2] # [1, 3]
mylist[1:3] # [3, 5]
mylist[3:] # [7, 9]
```

FUNCIONES BÁSICAS

Índices y slicing:

Acceso a subcomponentes de no scalares (list, dict, etc.).

```
myword = 'Hola Pedro'
myword[2] # 'l'
myword[5:] # 'Pedro'
myword[:4] # 'Hola'
```

FUNCIONES BÁSICAS

print:

Envio información como salida estándar (standard output).

```
myword = 'Hola Pancho'
print myword # Hola Pancho

mylist = [1, 2, 3, 4]
print mylist # [1, 2, 3, 4]
```

OPERADORES

Elementos que realizan funciones bastante comunes.

Operador + objeto = expresión.

Expresiones => objeto.

```
1 + 1 # 2
'a' == 'a' # True
True and False # False
```

CLASES DE OPERADORES

1. Aritméticos
   - `+`, `-`, `*`, `/`, `%`, `**`.
   - Output es int o float.
2. Comparación
   - `==`, `!=`, `>`, `>=`, `<`, `<=`.
   - Output es bool.

VARIABLES

Nombres definidos por el usuario para objetos particulares.

```
myage = 88
age_2000 = myname - 18
myname = 'Pancho'
complete_name = myname + ' Villa'
```