

Herramientas Computacionales para la Astroinformática

Cristian A. Vega-Martínez (oficina: IMIP, Académicos 2)
Facundo A. Gómez



Fundamentos de Python

Definiciones, objetos, programación imperativa

Python - ideas clave

En lo sucesivo, supondremos que tienen experiencia **básica** en desarrollo de códigos en Python.

- Es un lenguaje **interpretado**:
Se puede utilizar directamente en un intérprete, mediante la creación de scripts, o con una interfaz tipo IDE.
- Su instalación (y librerías) suele estar autocontenida en un directorio específico (e.g. sistema, anaconda, dir. personal)
- Utiliza (por defecto) **tipado dinámico de datos** en las declaraciones.
Toda “*variable*” tiene asignado un tipo (`type(var)`).
- Incluye una variedad de tipos básicos: `bool`, `int`, `float`, `str`...
- En Python, todo es un **objeto**.

Declaraciones en Python

Cada declaración en Python (e.g: `pi = 3.14`) involucra la creación de:

- Una **instancia**: la concretización/materialización de lo declarado, incluyendo la reserva de memoria, según las características del *"tipo de dato"*.
(`float 3.14` en el ejemplo)
- Una **referencia**: el identificador/nombre que apunta a una instancia específica. Esta no es equivalente a la instancia.
(`pi` en el ejemplo)

Python en todo momento lleva registro de la **lista de referencias** accesible desde el punto de ejecución o desde un objeto:

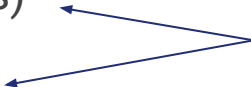
- esta varía según el flujo del código, y
- es accesible con `dir()` o por autocompletado (modo interactivo).

La clave para dominar Python:

¿Qué son los objetos?

Los lenguajes con orientación a objetos **generalizan** el concepto *tipo de dato*, permitiendo la creación de nuevas estructuras complejas llamadas **clases**.

Las clases, así como los tipos de datos, determinan cómo se comportarán sus instancias dentro del programa. Para ello, **cada clase puede contener**:

- **Atributos** (componentes)
 - **Métodos** (funciones)
 - Definición de uso de **operadores** (e.g: +, -, *, [], etc..)
- Accesibles mediante el operador punto.*
- 

Un **objeto** es una **instancia** de una **clase** definida.

Objetos - ideas clave.



- Una referencia **no es equivalente** a una instancia.
- Las referencias (identificador) **apuntan** a objetos (instancias).
- Atributos y métodos de los objetos son accesibles con el **punto** . desde sus referencias.
- Cada objeto tiene asociado un tipo: la **clase** de la instancia.
- Cada objeto tiene un identificador (**id**()), basado en la dirección de memoria donde se almacena.

Tipos de datos compuestos

Python incorpora varias clases definidas para crear objetos que almacenan otros objetos, siguiendo diferentes estrategias:

- **list**: listas mutables y ordenadas de objetos heterogéneos.
- **tuple**: agrupaciones inmutables de objetos, heterogéneos.
- **set**: agrupación no ordenada de elementos únicos.
- **dict**: diccionarios de objetos, indexados por etiquetas.

El copiado de listas, sets y diccionarios debe ser **explícito**, en caso contrario se asigna una nueva referencia al mismo.

Control de flujo

Algunas herramientas comunes de Python para el control de flujo:

- **Condicionales:** `if(...):` , `elif (...):` y `else:`
- **Loops:** `while(...):` y `for ... in ...:`
 - funciones generadoras (e.g. `range(...)`)
 - sentencias `enumerate(...)` y `zip(...)`
 - sentencias `break` y `continue`
- **Funciones:** `def fname(...):`
- Manejo de **errores:** `try:` y `except:`



Herramientas de I/O (input/output)

- stdout/stdin: funciones `print(...)` e `input(...)`
- Formato de **strings**: concatenación, *f*-strings y `str.format()`
- Argumentos de entrada en scripts: `from sys import argv`
- Lectura/escritura de archivos: `f = open("+", fname)`



Librerías de Python

Paquetes que se cargan utilizando **import** y que definen objetos útiles.

Algunos ejemplos usados en análisis de datos:

- **sys** - Acceso a parámetros y funciones específicos de sistema (argv, ps1)
 - **os** - Interacción con el sistema operativo
 - **time** - Acceso a hora y conversiones
 - **math, cmath** - Funciones matemáticas (y para números complejos).
 - **decimal, fractions** - Aritmética decimal de coma tanto fija como flotante, y número racionales.
 - **statistics** - Funciones de estadística matemática.
- **random** - Generar números pseudoaleatorios
 - **numpy** - Vectores y matrices multidimensionales
 - **matplotlib** - Visualización de datos
 - **scipy** - Cálculos científicos y técnicos
 - **pandas** - Datos Tabulados