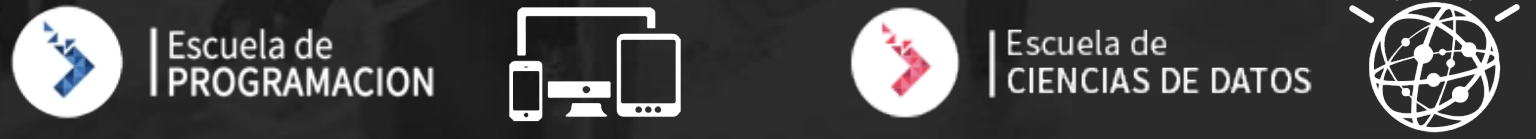
Introducción a la programación con Python



EANT | Escuela Argentina de nuevas tecnologías

Sobre la EANT

{Barro}

{Entrenar}

{Iralhueso}

{Explorar}

2 Reglas

1. Concepto nuevo? → Lo anoto → Lo googleo

2. No entendí? → Pregunto / Pregunto / Pregunto

Introducción a la Programación con Python

1. Fundamentos de Programación

-Resolviendo problemas a través de la programación
-Introducción a los sistemas
-Elementos de programación: variables y valores
-Poniéndonos en el lugar de un robot

Programación

Qué?
Por qué?
Para qué?

Programación

Ó...cómo entrenar a tu ROBOT

Programar es como entrenar a tu propio robot para que resuelva problemas

La preguntas es...

...cómo hablar con tu robot?
...cómo piensa tu robot?

Tenemos que explicarle muchas cosas a nuestro Robot antes de comenzar!

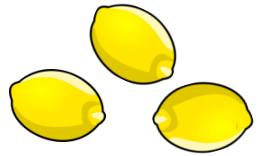
Programación

Sistemas para resolver el mundo

En principio se debe definir una manera en que hablaremos con nuestro Robot sobre los ELEMENTOS del problema

Por ejemplo:
Necesito que mi ROBOT traslade 620 limones de un almacén A a un almacén B pero lo tiene que hacer mediante un canasto que sólo permite llevar de a 50 limones por vez.

Los Elementos	Los Valores
Limones en Almacén A	Tiene 620
Limones en Almacén B	Tiene 0
Limones en el Canasto	Puede llevar hasta 50



Cantidades de Limones

Primero le voy a enseñar a mi Robot que estos son los **elementos** del **sistema**

Y luego…que lo que importa de estos elementos es la **cantidad de limones** que tienen o pueden tener

Programación

Cada cosa por su nombre

Elementos

Limones en Almacén A

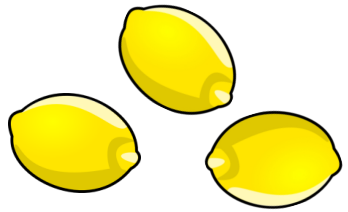
Limones en Almacén B

Limones en el Canasto

Para representar los elementos de un sistema en programación se utilizan **VARIABLES**

Los nombres que le demos a las variables deben describir lo mejor posible lo que representan

Valores



Unidades de Limones

El tipo de valor que se utiliza en cada variable debe asociarse a un **TIPO DE DATO**

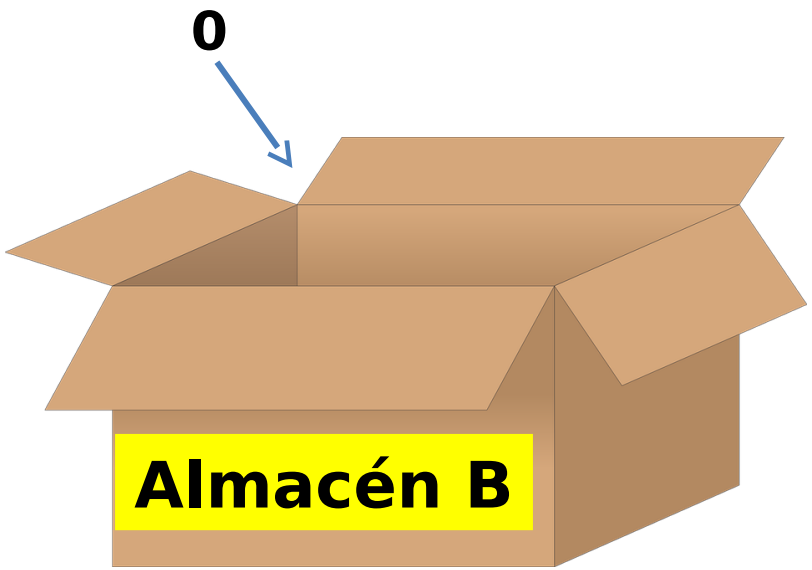
Podemos elegir entre muchos tipos de datos:

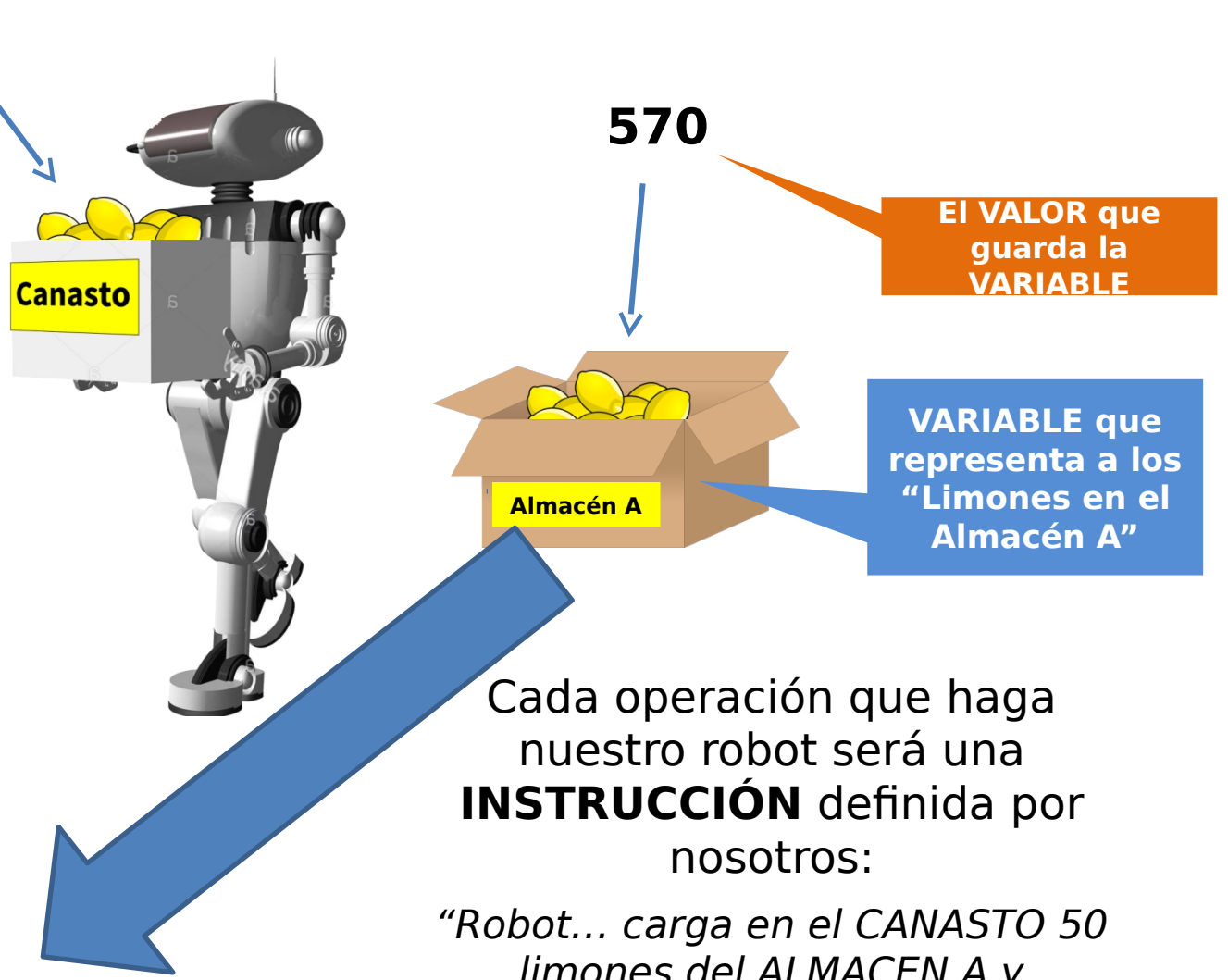
- Numéricos
- De texto
- Lógicos (Verdadero / Falso)

Programación

Variables e Instrucciones

Podemos pensar a las **VARIABLES** como “cajas etiquetadas” que nos permitirán representar y manipular a cada uno de los elementos y en donde guardaremos el **VALOR** que caracteriza a dicho elemento en cada instante.





Cada operación que haga nuestro robot será una **INSTRUCCIÓN** definida por nosotros:

“Robot... carga en el CANASTO 50 limones del ALMACEN A y descargalo en el ALMACEN B”

Programación

Pensando detalladamente en código

Para enseñar a nuestro ROBOT deberemos ser muy detallados sobre cada paso de la operación

"Robot… carga en el CANASTO 50 limones del ALMACEN A y descargalo en el ALMACEN B"

Cómo lo pienso Yo…	Cómo lo piensa mi Robot…
1) Primero definir cual es la situación de cada elemento al comienzo	1) >> almacen_A = 620 >> almacen_B = 0 >> canasto = 0
2) Tomar 50 Limones del Almacén A y ponerlos en el Canasto	2) >> canasto = 50 >> almacen_A = almacen_A – canasto
3) Descargar el Canasto en el Almacén B	3) >> almacen_B = almacen_B + canasto >> canasto = 0

Programación

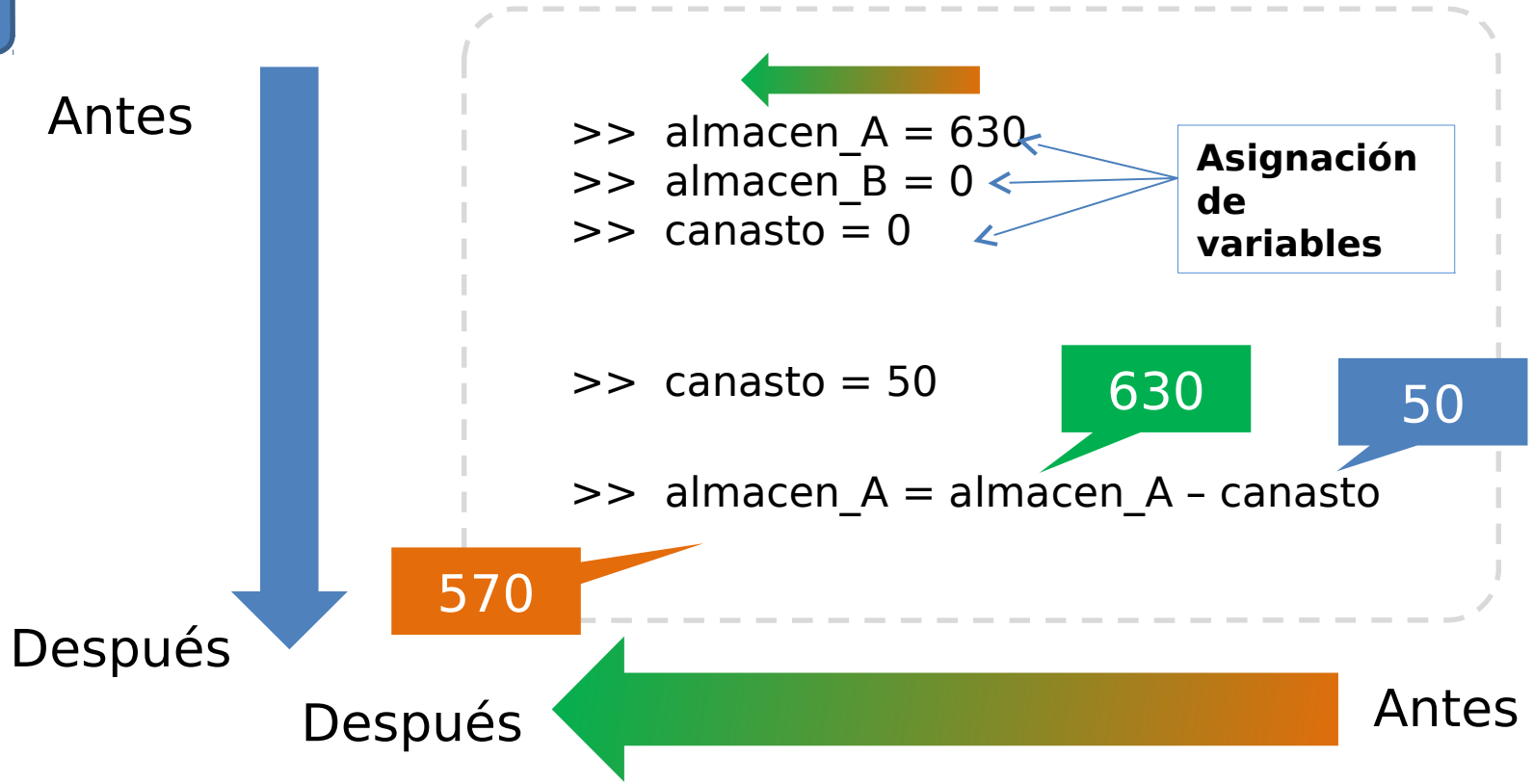
Algo de terminología

Cómo piensa mi Robot...

El código se interpreta leyéndolo :

1) de Arriba a Abajo y..

2) de Derecha a Izquierda

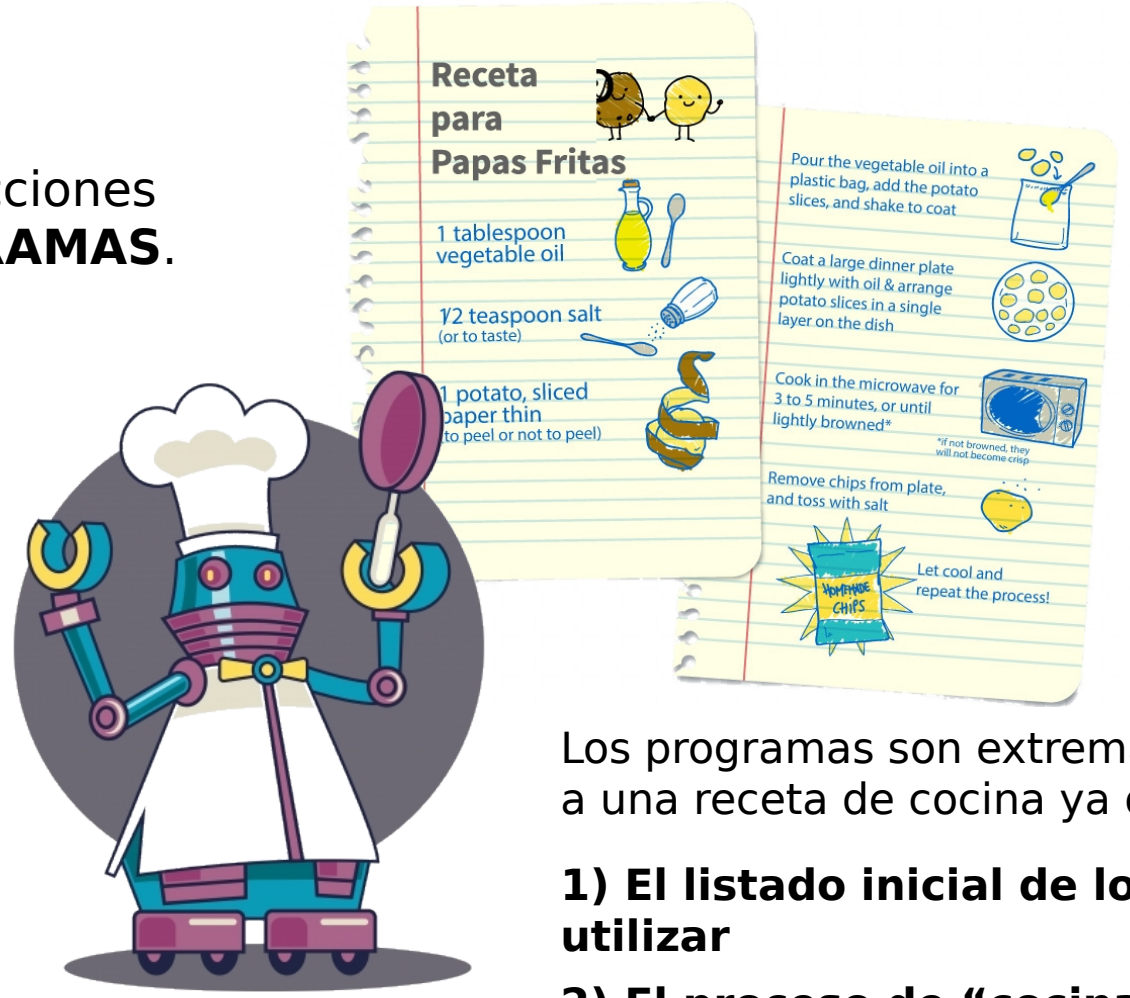


Programación

Programas

Las agrupación de muchas instrucciones son lo que dan lugar a los **PROGRAMAS**.

Un PROGRAMA es una receta del “paso a paso” que debe seguir el Robot en la utilización, mezcla y transformación de los elementos para dar un resultado final.



Los programas son extremadamente parecidos a una receta de cocina ya que siempre tienen:

1) El listado inicial de los ingredientes a utilizar

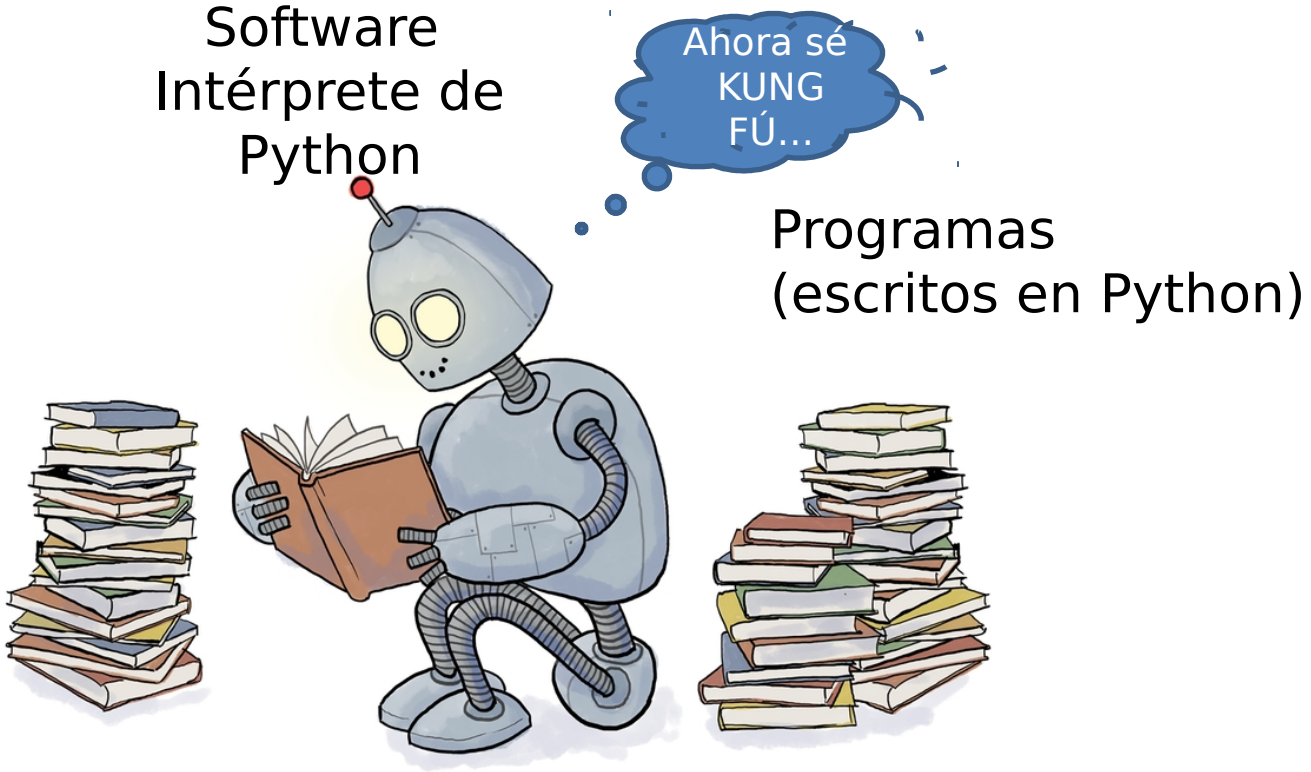
2) El proceso de “cocinado” de los ingredientes

3) Un resultado final

Programación
Robots Todo-Terreno

Mediante los programas de software podemos hacer que un mismo “Motor de Software” (el Robot) se alimente con distintos “Programas” (Recetas) para la realización de distintos tipos de tareas según se requiera.

Los programas se guardan en el disco rígido como archivos con la extensión del tipo de lenguaje con que se haya desarrollado (Python .py)



Programación

Interfaces: charlando con la máquina

Muchas veces vamos a necesitar “charlar” con nuestro robot, entregándole información (INPUTS) y a su vez necesitando información por parte de éste (OUTPUTS).

Las **Interfaces** (software destinado a ejecutar la interacción entre usuarios y máquinas) pueden ser tan sofisticadas como se requiera, desde un mega portal web hasta una simple terminal de ingreso y salida de datos.

Hablemos de software

Poniéndonos un poco más técnicos

Usuarios

Interfaz

La “terminal”
(Software de interfaz)

Inputs

Outputs

Ejecución

Motor de Software
(Intérprete de Python)

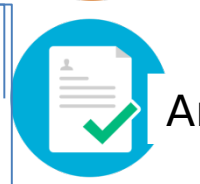
Edición del Programa
(Desarrollo)

Código de Programa
(Archivos .py)

Recursos

Archivos HTML/CSS

Archivos de imagen/audio



Archivos de texto

Archivos de Código

Bases de Datos

Introducción a la Programación con Python

2. Python basics

-Bienvenidos a Python!
-Descarga e instalación del Entorno de Desarrollo
-Exploración del IDLE
-Primer código en Python
-Variables y tipos de datos
-Uso de la consola

Python
Por qué Python?

Pensado para la enseñanza académica

Gran Comunidad de Desarrollo y Colaboración

Especialmente diseñado para el trabajo con Datos

python™

Gran cantidad de librerías de código disponibles

Gran capacidad para el desarrollo de software en general

Python
Entorno de Trabajo

Qué se instaló en mi computadora?

Motor de Python

Es el “Robot”, quién va a interpretar y ejecutar nuestros códigos

IDE

Es el “Entorno de Desarrollo” con el que crearemos nuestros programas

Terminal

Es la Interfaz mediante la cual dialogaremos con nuestros programas cuando esté ejecutándose

Librerías

Python ya viene con una gran cantidad de códigos pre-hechos muy útiles para hacer cosas complicadas

Pip

En internet hay muchos más códigos “útiles”. Este software nos permite bajarnoslos muy fácilmente

Práctica 1

De lleno con Python

-Exploración de Impares.py
-Asimilación del lenguaje
-Qué tan entendible es Python?

Variables

Tipos de datos

Strings (Texto)

```
1  nombre = ' Jorge'
2  pronostico = "Vá a llover mañana!"
3  url = 'http://hablemosdecodigo.com'
```

Los tipos de datos de texto deben ser identificados envolviéndose en " o ' ya que de otra forma Python los confundiría con variables

Integers (Números enteros)

```
1  a = 3
2  b = 4
3  hipotenusa = 5
```

Floats (Números decimales)

```
1  a = 3.0
2  b = 4.0
3  hipotenusa = 5.0
```

Operaciones aritméticas

```
>>> 5 + 6
>>> 0 - 99
>>> 5.0 / 2.0
>>> 5 / 2
>>> 4 * (7 - 2)
```

Las variables de Python se asignan dinámicamente, es decir que toman el tipo del valor asignado.

Variables

Buenas prácticas para los nombres

- **Seleccionar siempre nombres que identifiquen claramente lo que la variable representa**

Ej – Malo: dato vs Ej – Bueno: almacen_limones

- **Usar siempre minusculas:**

Ej – Malo: Almacen_Limones vs Ej – Bueno: almacen_limones

- **Palabras compuestas**
  - **Utilizar Camel Case:**

Ej: almacenLimones ó (si necesito más letras)

almacenLimonesDulcesQueVencieronPeroIgualNosGustan

  - **Utilizar simple_text:**

Ej: almacen_limones

- **No comenzar un nombre con un número**

Un buen nombrado de las variables ayuda luego a poder entender nuestro código cuando éste se hace muy complejo y además para que otros lo entiendan también

Práctica 2

Variables, tipos de datos e interfaces

-Jugando con Variables y Tipos de datos
-Utilización de interfaces de entrada y salida

Trivia

1. Cuál es la diferencia entre:

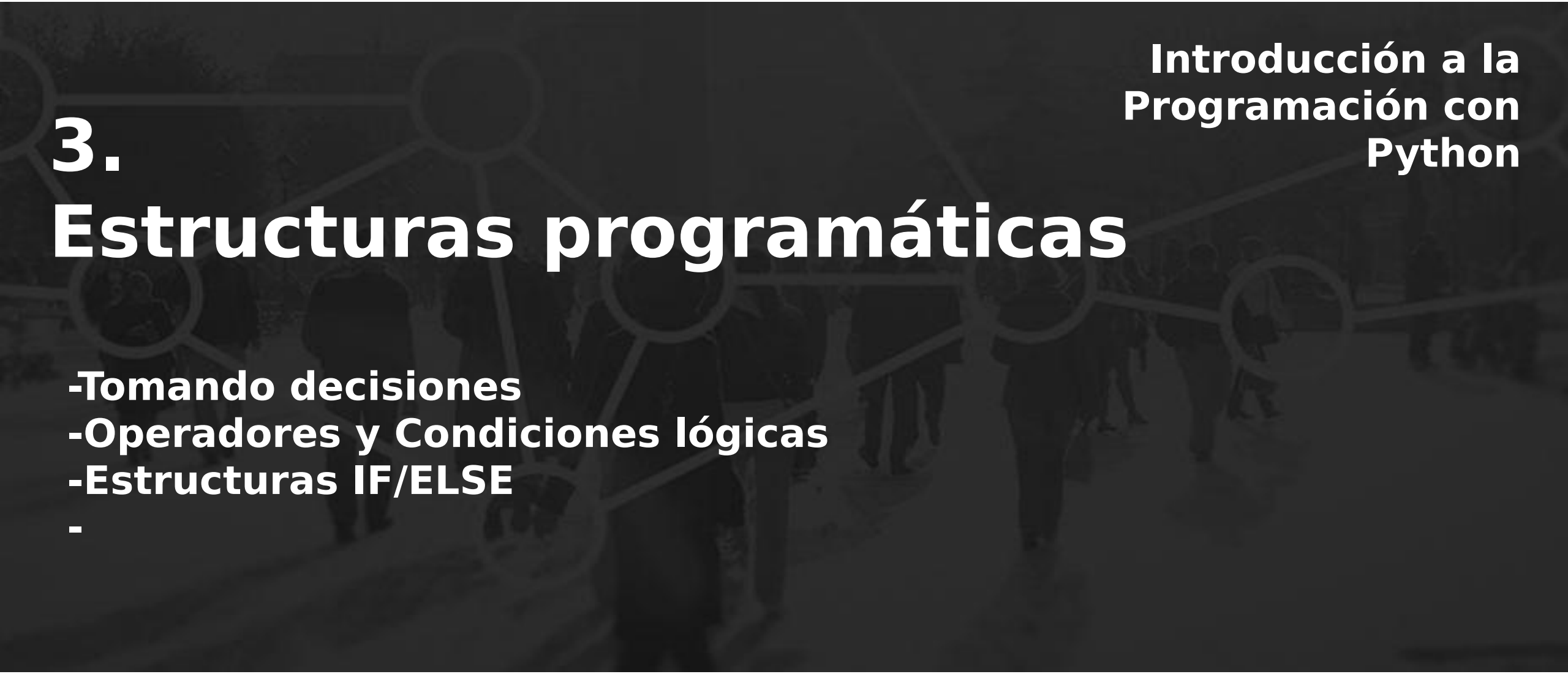
   mi_variable = 5 y mi_variable = '5'?

2. Cuál es la diferencia entre print(n) and print('n')?

   Intentá ingresar en la consola:

   ```
   n = 5
   print(n)
   print('n')
   ```

3. Intentá volver a escribir hola_mundo.py sin usar una variable

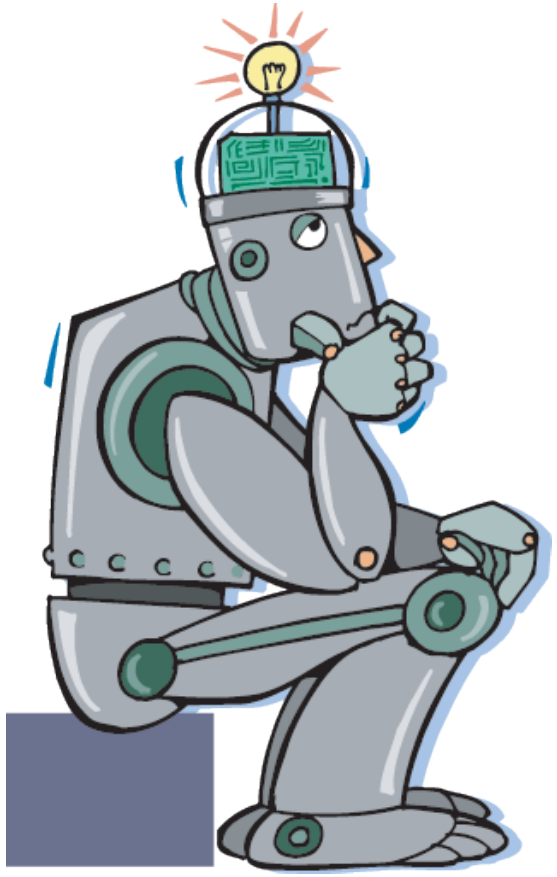


Introducción a la Programación con Python

3. Estructuras programáticas

-Tomando decisiones
-Operadores y Condiciones lógicas
-Estructuras IF/ELSE
-

Tomando Decisiones



Tomando decisiones

Construcción lógica de una decisión

Tenes que tomar una gran decisión mañana: tomar el autobús o caminar.

Si llueve, entonces tomarás el autobús; de lo contrario, caminarás.

1. Primero, hay algo que es **verdadero** o **falso**. En este caso, lo que es verdadero o falso es la presencia o ausencia de lluvia. Esto se llama una **condición**.
2. Luego, se realiza una acción distinta dependiendo de si la condición es verdadera o falsa:

Si está lloviendo, entonces tomas el autobús. **Si no está lloviendo, entonces caminás**

Las computadoras procesan las decisiones de la misma manera. En el código de la computadora, podemos darle a la computadora una condición para evaluar y las acciones a tomar si esa condición resulta verdadera o falsa. Este concepto se llama **ramificación** porque el código puede "ramificarse" en dos o más direcciones cuando lo necesitamos.

Lógica Booleana

Expresándonos lógicamente

Toda declaración puede ser VERDADERA ó FALSA

- Mi edad es 30
- Mi nombre no es Manuel
- 1 es mayor que 100
- 1 es menor que 100

En programación podemos formular estas declaraciones de la siguiente manera:

- edad == 30
- nombre != “Manuel”
- 1 > 100
- 1 < 100

Estas expresiones debemos leerlas siempre en forma de pregunta (?)

Una declaración verdadera tiene por resultado un valor booleano

True

Una declaración falsa tiene por resultado un valor booleano

False

Operadores lógicos

Operador	Tipo	Propósito
=	Asignación	Asigna un valor a una variable
==	Comparación	Chequea si 2 valores son iguales
!=	Comparación	Chequea si 2 valores NO son iguales
>	Comparación	Mayor qué...
>=	Comparación	Mayor o Igual qué...
<	Comparación	Menor qué...
<=	Comparación	Menor o Igual qué...

Tomando decisiones: if

1

```
n = 50
if n < 100:
    print(“La condición es verdadera!")
```

2

```
clima = “Hoy está lloviendo”
if clima==“Hoy está lloviendo”:
    print(“Hay que tomar el colectivo!")
else:
    print (“Vamos caminando que esta lindo!")
```

3

```
peso = 85
if peso<78:
    print(“Tenés que comer!")
elif peso==78:
    print (“Estás en el peso justo!")
else:
    print(“Cortame los postres...”)
```

Anidaciones de código

```
if dia== ‘Sábado’:
        print(‘Hoy se saleee….fuerte’)
elif dia==‘’Domingo’:
        if mi_condicion=‘jaqueca’:
                print(‘Recobrarse, luego descansar’)
        else:
                print (“Descansar")
else:
        print(“Trabajar, trabajar, trabajar”)
```

“bloque de código”

Estos también son “bloque de código” pero dentro de otro “bloque” de mayor jerarquía/nivel

Nivel 0

Nivel 1

Nivel 2

La **“indentación”** es la manera en que Python reconoce las distintas jerarquías de código

Expresiones combinadas

Estos operadores sirven para combinar más de una condición

```
if a == 3 and b == 4:
    print("La hipotensua es 5.")

if a == 3 or b == 4:
    print("La hipotenusa podría ser 5.")
```

Podés usar tantos de estos como necesites, pero cuando comenzar a combinar operadores a veces tenés que incluir paréntesis para especificar el orden en que se evalúan las condiciones.

Intentá escribir esto en el intérprete:

```
>>> 1 == 100 and 1 == 2 or 1 == 1
>>> (1 == 100 and 1 == 2) or 1 == 1
>>> 1 == 100 and (1 == 2 or 1 == 1)
```

Probá éste código de condiciones combinadas:

```
if (a == 3 and b == 4) or (a == 4
    and b == 3):          print("La
    hipotenusa es 5.")
```

Operadores Booleanos

Tablas de Verdad

A	B	A and B	A or B	!(A and B)	!(A or B)
V	V	V	V	F	F
V	F	F	V	V	F
F	V	F	V	V	F
F	F	F	F	V	V

Trivia

1. Cuál es la diferencia entre n = y ==?

2. Crear un programa edad.py que le pregunte a los usuarios su edad y luego le imprima las cosas que puede hacer según su edad (Por ejemplo: votar / comprar alcohol).
   NOTA: la función nativa **int()** puede convertir datos de tipo String (texto) en datos de Tipo Integer (Números enteros)

Funciones

En programación, una función es un bloque de código con un nombre

Ejemplos de funciones nativas

```
print("Hola Mundo!")
```

Funciones que "hacen"

```
numero = int("22")
```

Funciones que "entregan"

Las funciones son importantes porque:

1) Nos facilitan el agrupar una serie de operaciones complejas bajo un elemento simple
2) Nos permiten crear porciones de código reutilizables
3) Nos permiten escribir un programa de forma organizada y legible

Construyendo una función propia

Primero siempre es necesario declarar nuestra Función….

Operador Python para declarar una función

Nombre que identifica a la función

OPCIONAL: parámetros que requiere la función

```
def nombreFuncion (parametro1,  parametro2, ...):
    #Código
    #Código
    #Código
    return resultado
```

Acá va el código que queremos reutilizar mediante la función y que utilizaría a los **parámetros** ingresados si estos fueron definidos

OPCIONAL: operador Python para el retorno de valores

OPCIONAL: el valor a retornar

…luego para usar la función en nuestro programa

```
>>> nombreFuncion (parametro1, parametro2, ….)
```

En modo de ejecución a los **parámetros** se los suele llamar **argumentos**

Qué es un “método”?

Un **método** es una **función** asociada a un **objeto**

Nada mejor que un buen ejemplo:

```
>>> lugar = “la casa rosada"
>>> lugar_importante = lugar.title()
>>> print(lugar_importante)
```

La variable lugar es un objeto de tipo String, y por lo tanto tiene varios métodos asociados para facilitarnos las cosas

Nos damos cuenta de que se trata de un método siempre que vemos un “.” y un “()”:

objeto.metodo()

Ojo al piojo: En Python…TODO es un OBJETO

Listas

Las listas son variables especiales que se usan para agrupar a más de un valor

alumnos = [‘Juan', 'Jorge', ‘Diego', ‘Tobias‘]

elementos = [12, ‘Auto', ‘Negro', ’23.5‘]

Hay dos características definitorias de una lista:

- Son ordenadas: La secuencia en la que se agregan cosas a una lista se conserva.
- Pueden contener duplicados.

Listas

Operadores básicos

Agregar elementos a una lista

```
>>> mi_lista = ['A','B','C']
>>> mi_lista.append('D‘)
>>> print(mi_lista)
```

Contando elementos a una lista

```
>>> mi_lista = ['A','B','C']
>>> len(mi_lista)
>>> 3
```

Retornando un elemento/item de una lista

mi_lista = ['A','B','C', ‘D']

index	0	1	2	3
item	A	B	C	D

Para tomar el primer item:

```
>>> mi_lista[0]
```

Para tomar el último:

```
>>> mi_lista = ['A', 'B', 'C', 'D']
>>> ultimo = len(mi_lista) - 1
>>> mi_lista[ultimo]
```

Listas

Operadores básicos

Verificando si existe un elemento en una lista

```
>>> 2 in [1, 2, 3]
True
>>> 5 in [1, 2, 3]
False
>>> 'A' in ['A', 'B', 'C']
True
```

Retornando la posición de un elemento

```
>>> mi_lista = [‘Juan', 'Jorge', ‘Diego', ‘Tobias‘] >>> mi_lista.index(‘Diego')
2

>>> mi_lista = [‘Juan', 'Juan', ‘Juan’]
>>> mi_lista.index(‘Juan')
0
```

¿Qué pasa si buscamos un elemento que no está en la lista?

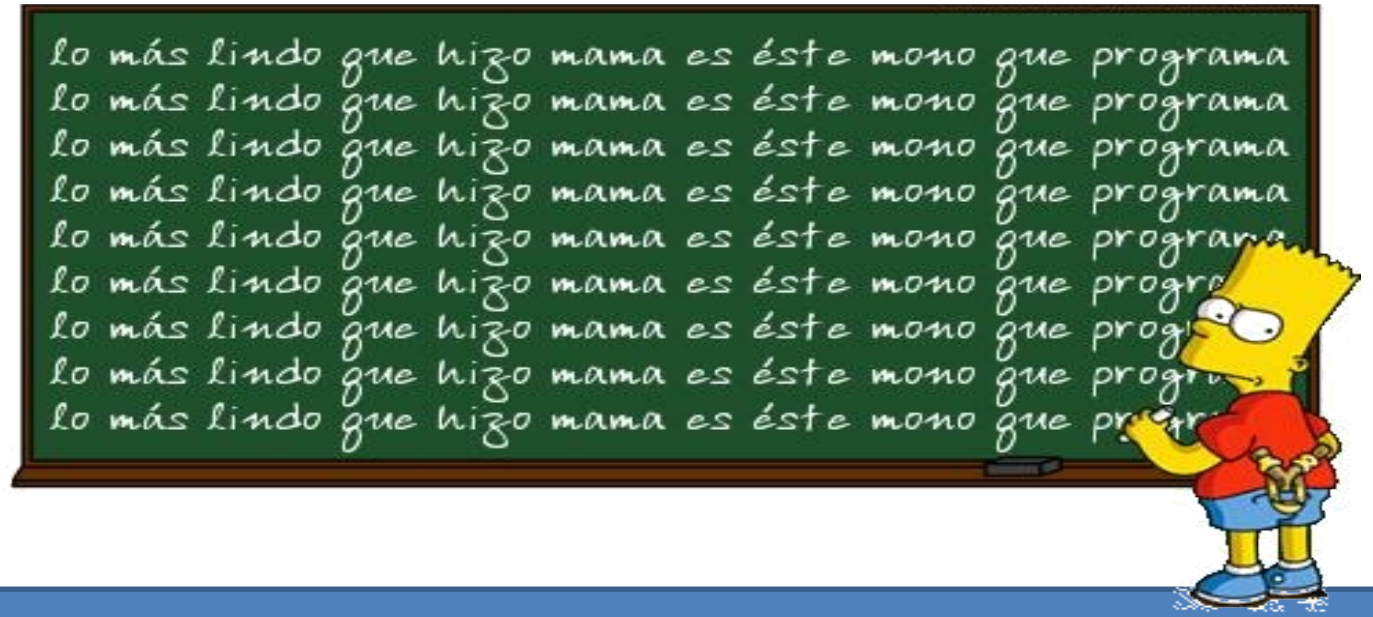
Trivia

1. ¿Qué dos características hacen que una colección sea una lista?
2. Escriba un código que permita a los usuarios ingresar sus tres comidas favoritas. Almacene esos alimentos en una lista.
3. Imprima el elemento del medio de esta lista usando un índice: ['Mercurio', 'Venus', ‘Tierra'].

   ¿Podrías cambiar tu código para trabajar con una lista de cualquier tamaño (suponiendo que haya un número impar de elementos)?
4. ¿Qué sucede cuando ejecutas este código? ¿Sabes por qué?

   ```
   >>> mi_lista = ['A', 'B', 'C']
   >>> mi_lista[len (mi_lista)]
   ```

Loops



Loops = Bucles ó Iteraciones

El poder real de las computadoras está en su capacidad para ejecutar tareas repetitivas sin quejas.

Cuando queremos que un programa de computadora ejecute la misma pieza de código varias veces, ajustamos ese código dentro de un **bucle**.

While Loop

"Hace algo hasta que yo diga basta"

Estructura:

```
while (condicion es Verdadera):
    #Código
```

Ejemplo:

```
favoritos = []
mas_items = True
while mas_items:
    user_input = input("Ingresá algo que te gusta: ")
    if user_input == '':
        mas_items = False
    else:
        favoritos.append(user_input)

print("Estas son todas las cosas que te gustan!")
print(favoritos)
```

For-Each Loop

"Hace algo N número de veces"

Estructura:

```
for  item in lista :
    #Código
```

Ejemplo:

favoritos = ['milanesa','ravioles','vacio como venga']

```
for comida in favoritos :
    print("Me gusta " + comida)
```

Probar:

```
>>> for i in [1, 2, 3]:
    print(i)
```

```
>>> for ch in "Hola!":
    print(ch)
```

Rangos

Abrir un shell de Python y probar estas sentencias:

```
>>> list(range(5))
>>> list(range(3,7))
>>> list(range(7,3))
>>> list(range(-2,2))
```

La función **list()** fuerza el rango a una lista que podemos leer fácilmente.

range () nos da una colección de números

La función range() es muy útil para controlar For Each Loops:

Cosas simples...

```
for i in range(5):
    print( str(i) + " vueltas!" )
```

..o un poco más complejas...

```
def imprimir(lista_super):
    for i in range(len(lista_super)):
        print(str(i + 1) + ". " + str(lista_super[i]))
```

Librerías y Funciones Nativas

La biblioteca estándar de Python es muy rica y proporciona muchos códigos reutilizables

Importar a nuestro código la función desde el módulo que la contiene...

Módulo

Función o Método del Módulo

```
from os import getcwd

dnd_estoy = getcwd()
```

...luego ejecutar la función

También puedo importar al objeto completo y luego utilizar todas sus funciones:

Método

Propiedad

```
>>> import os
>>> os.getcwd()
>>> os.environ
>>> os.getenv('HOME')
```

Existen muchos módulos útiles:

```
>>> import datetime
>>> datetime.date.today()
>>> datetime.date.today().day
>>> datetime.date.today().month
>>> datetime.date.today().year

>>> import time
>>> time.strftime("%H:%M“)
```

```
>>> import sys
>>> sys.platform
>>> print(sys.version)
```

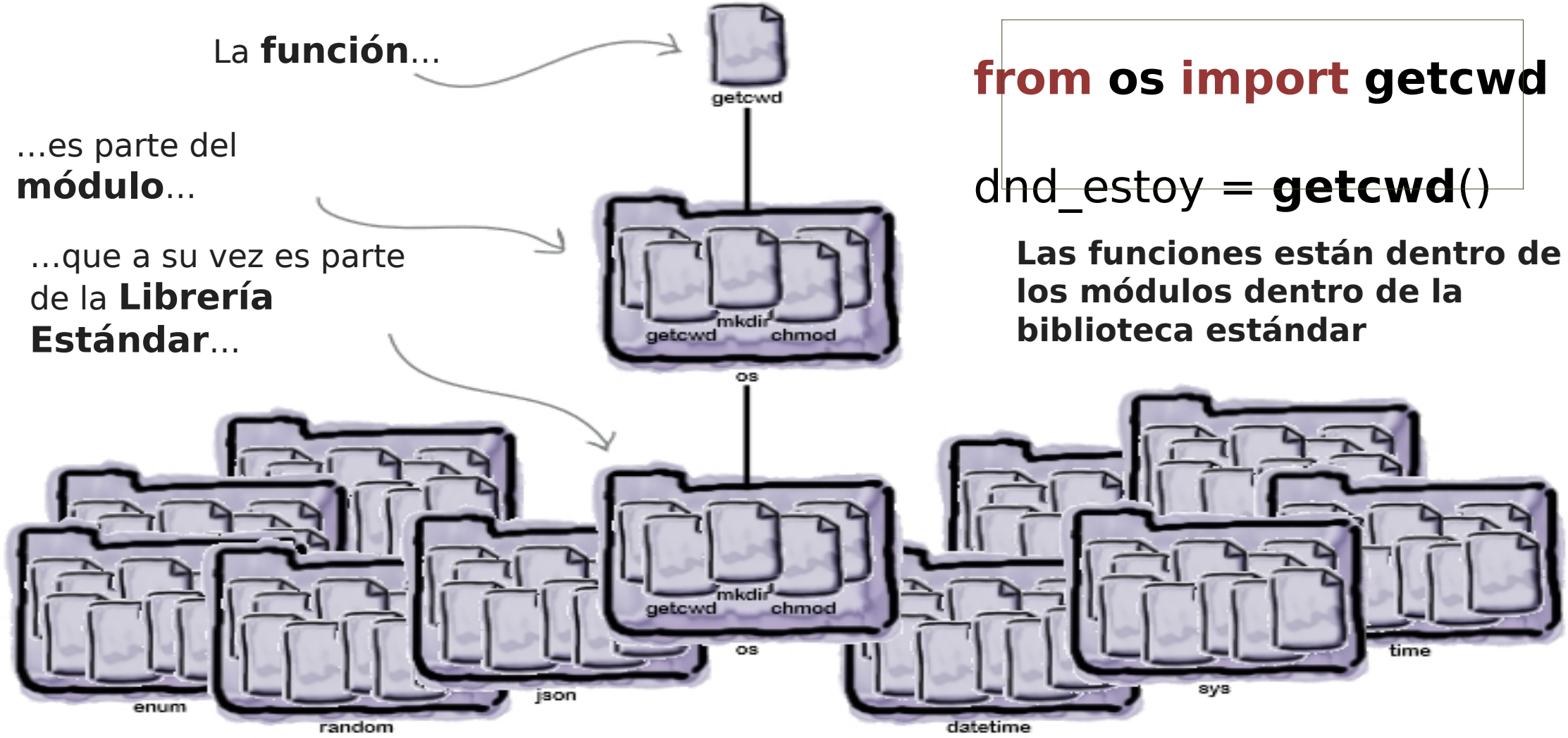
La función help() nos ayuda a explorar las librerías

Biblioteca Estándar de Python

La **función**...

...es parte del **módulo**...

...que a su vez es parte de la **Librería Estándar**...




```
from os import getcwd

dnd_estoy = getcwd()
```

Las funciones están dentro de los módulos dentro de la biblioteca estándar

Librerías para Python

Información!

1

La documentación de Python tiene todas las respuestas sobre la biblioteca estándar.

Aquí está el punto de partida:
https://docs.python.org/3/library/index.html

2

La biblioteca estándar no es el único lugar en el que vas a encontrar excelentes módulos importables para usar en tu código. La comunidad de Python también es compatible con una próspera colección de módulos de terceros, algunos de los cuales exploraremos más adelante.

Explorá el repositorio desarrollado por la comunidad:
http://pypi.python.org.

Explorar y Jugar es la clave

Guías y consejos!

Python es un lenguaje diseñado especialmente para que probar, explorar y aprender cosas nuevas:

1) **Usar mucho la Consola REPL(>>>) para probar objetos, comandos y todo lo que tengas dudas**

2) **Acostumbrate a usar la instrucción help() para obtener info sobre módulos, objetos, funciones, comandos que no sabés bien cómo se usan**

Ejemplo: >>>help(range)

3) Usá la función dir() para estudiar objetos (métodos, atributos, etc)

Ejemplo:
>>> import random
>>> dir(random)
>>> help(random.randint)