Fundamentales de Python

(cocinado con ♥ en Ludidactas)

Python es un lenguaje de programación.

Como todo lenguaje, tiene una sintaxis, una estructura que hay que respetar.

palabras clave — identificadores

```
for nombre in ['Lala', 'Lolo', 'Pepe']:
    print('Hola', nombre)
```

tabulación — marcadores sintácticos

Quizás la distinción más importante a hacerse sea entre *instrucciones* y *expresiones*.

Una **instrucción** es imperativa: describe una *acción*, y como consecuencia tiene un ***efecto***.

Una **expresión** resuelve a un ***valor***. Es como si reescribiéramos la expresión con su resultado. Su **tipo** es el tipo de dato al que resuelve.

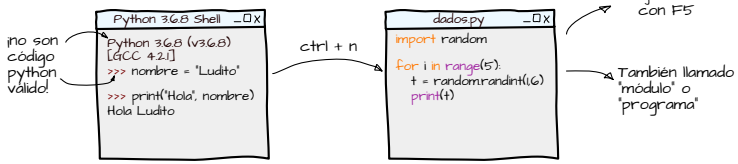
integer — float

```
n = len('Hola') + 0.5 * 3
```

La expresión resuelve a float — *precedencia!

Esta instrucción (asignación) tiene el efecto de guardar el valor de la expresión 5.5 (float) en el identificador (variable) n

Desde www.python.org podemos descargar el instalador, que incluye un intérprete y un editor de scripts.



En el **intérprete** probamos código línea por línea. En un **script** escribimos muchas líneas que se ejecutan en secuencia.

Instrucciones

while for def if import else return

La estructura de las instrucciones se expresa principalmente en *subordinaciones:*

Todo el código que vemos tabulado al mismo nivel, está subordinado a una misma instrucción.

```
for nombre in ['Lala', 'Lolo', 'Peperino']:
    if len(nombre) > 7:
        print('Qué nombre largo', nombre)
    else:
        print('Holaaa', nombre)
```

subordinado — subordinado — subordinado

tabs

Condicional - if/else

Bifurca la ejecución del código basado en una condición: "si se cumple tal condición, hacé esto; sino, hacé esto otro".

```
nombre = input()
if nombre == 'Peperino':
    print('Muchas gracias por el vino')
else:
    print('¿Cómo estáis? ¿Cómo os sentáis?')
```

La expresión entre "if" y los dos puntos tiene que resolver a bool (True/False)

La "rama" del else es opcional

¡la tabulación indica el interior de las "ramas"!

Repeticiones - for/while

Repite un conjunto de instrucciones basado en un conjunto de datos (for ... in ...:) o en una condición (while ...:)

```
nombres = ['Chisi', 'Pata', 'Pu']
for nombre in nombres:
    print('Oa', nombre)
```

Este identificador tiene que apuntar a una secuencia existente

list, dict, range, etc

Este identificador se elige entre el for y el in, y dentro del código repetido está disponible conteniendo cada uno de los elementos de la secuencia que hayamos puesto entre el in y los dos puntos

Al igual que en if, esto tiene que resolver a bool

```
gnoccis = 40
while gnoccis > 0:
    pinchados = randint(1,3)
    print("Comiendo", pinchados)
    gnoccis = gnoccis - pinchados
print("Ñam")
```

Cada vez que se llega al final de las instrucciones subordinadas al while se vuelve a evaluar la condición, y si da True, se vuelven a ejecutar.

Módulos - import

Importa todas las funciones y variables de otro script. Hay unos cuantos que son standard y están automáticamente disponibles, como random, os y math

```
import random
print(random.randint(1,6))
```

Accedemos a los objetos del módulo a través del operador punto

```
from random import randint
print(randint(1,6))
```

Alternativamente, podemos importar sólo los objetos que necesitemos

Funciones - def/return

Agrupa instrucciones y provee una interfaz para invocarlas.

```
def circulo():
    for i in range(24):
        forward(10)
        right(15)
```

Definición

Invocación

```
circulo()
```

Es como si esto se reescribiera y reemplazara por el contenido que definimos arriba

La **definición** de una función es una instrucción, y tan solo tiene por **efecto** hacer que exista.

La **invocación** de una función es una expresión, resulta en un **valor**.

Podemos definir el valor al que va a resolver la invocación con **return.** Si omitimos el return, el valor va a ser **None**.

Las funciones pueden también **recibir** valores al momento de invocarse.

Esos valores se guardan dentro de variables que existen sólo dentro de la función, que llamamos **parámetros**. Los declaramos entre paréntesis y separados por comas, en la línea def.

Cada vez que se invoca a la función, podemos asignar valores distintos a esos parámetros. Los valores que le pasamos en una invocación se llaman **argumentos**.

Esto devuelve un float...

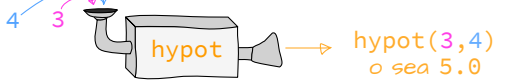
```
from math import sqrt
def hypot(cat1, cat2):
    d2 = cat1**2 + cat2**2
    return sqrt(d2)
```

...y es lo que la función retorna...

```
print("La distancia desde (0,0) hasta (3,4) es", hypot(3,4))
```

...así que esto resuelve a float

Son como maquinitas...



Anidación

El potencial de estas instrucciones está en la posibilidad de utilizarlas unas dentro de otras. Llamamos a esto **anidar**.

Un def puede contener un if que contenga un for que contenga otro if. Etc.

En Python, la anidación queda expresada por las tabulaciones.