PYTHON PARA LINGÜISTAS

CONCEPTOS BÁSICOS DE PYTHON:

ESTRUCTURAS DE DECISIÓN, BUCLES, FUNCIONES

ALEJANDRO ARIZA

CENTRE DE LLENGUATGE I COMPUTACIÓ

UNIVERSITAT DE BARCELONA

¿QUÉ SABEMOS DE COMPUTACIÓN?

- Computación = Datos + Algoritmos
- Datos: “ingredientes del pastel”, “el pastel en sí”
- Algoritmo: un proceso – preparar un pastel con los ingredientes
- Un algoritmo tiene datos de entrada y, a menudo, datos de salida
- Un algoritmo puede contener múltiples pasos intermedios
- Para mantener controlados los cambios, utilizamos las variables

¿QUÉ SABEMOS DE TIPOS DE DATOS?

- Hay diferentes tipos de datos, cada uno con sus respectivas operaciones
- Tipos numéricos: Enteros (1,2,3); Flotantes (1.5, 3.14, 6.667)
- Tipo booleano: True, False
- Tipo string: “un”, “un string”, “un string más largo con símbolos #$%^”
- Tipo lista: [“contiene”, “otros”, “tipos”, “de datos”, 3, True]

¿QUÉ SABEMOS DE TRABAJAR CON DATOS?

- Hemos aprendido las operaciones básicas de diferentes tipos de datos
- El tipo de datos con el que vamos a operar delimita las operaciones que podemos ejecutar:

    Los tipos numéricos se pueden modificar usando operaciones algebraicas

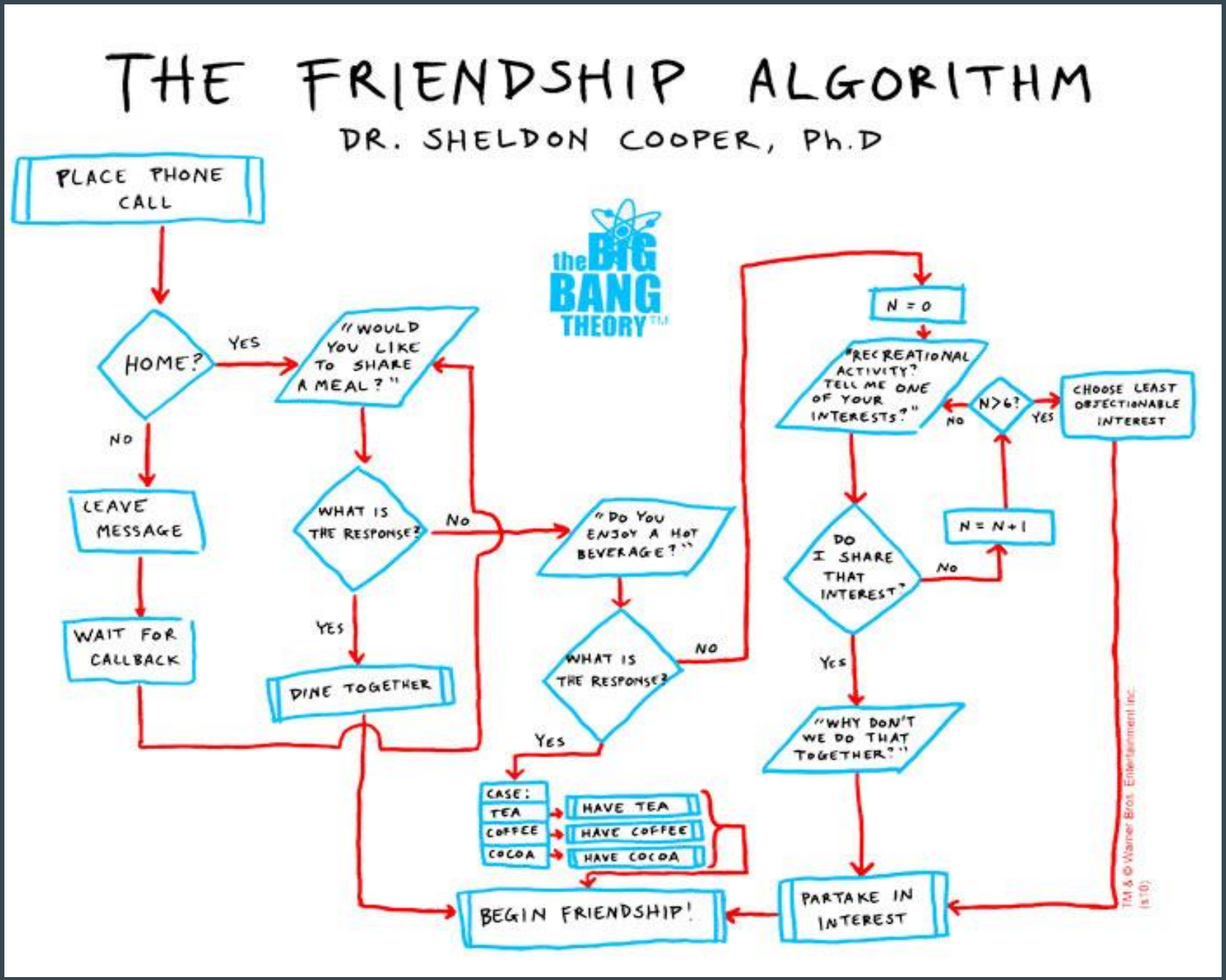
    Los strings y las listas pueden ser concatenados o segmentados

FROM SIMPLE OPERATIONS TO REAL PROGRAMS

- Knowing data types and their operations is important for programming
- There is more to programming than just data types and basic operations
- In this class we will learn about some of the most important and frequently used tools for programming: control structures, loops, and functions

DE OPERACIONES BÁSICAS A PROGRAMAS

ESTRUCTURAS DE CONTROL
CONDICIONES


THE FRIENDSHIP ALGORITHM
DR. SHELDON COOPER, Ph.D
PLACE PHONE CALL
HOME?
YES
NO
"WOULD YOU LIKE TO SHARE A MEAL?"
LEAVE MESSAGE
WAIT FOR CALLBACK
WHAT IS THE RESPONSE?
No
YES
DINE TOGETHER
"DO YOU ENJOY A HOT BEVERAGE?"
WHAT IS THE RESPONSE?
NO
Yes
CASE:
TEA
COFFEE
COCOA
HAVE TEA
HAVE COFFEE
HAVE COCOA
BEGIN FRIENDSHIP!
N = 0
"RECREATIONAL ACTIVITY? TELL ME ONE OF YOUR INTERESTS."
N>6?
NO
YES
CHOOSE LEAST OBJECTIONABLE INTEREST
DO I SHARE THAT INTEREST?
No
N = N+1
Yes
"WHY DON'T WE DO THAT TOGETHER?"
PARTAKE IN INTEREST
TM & © Warner Bros. Entertainment Inc. (s10)

ESTRUCTURAS DE DECISIÓN

- En un programa, se puede evaluar una o varias condiciones para determinar cuáles serán los próximos pasos a tomar:
  - “¿hay comida en el frigorífico?” – “sí” → ponernos a cocinar / ”no” → ir a comprar
  - “¿tiene batería nuestro móvil?” – “sí” → utilizarlo / ”no” → ponerlo a cargar
- Estas estructuras suponen un punto del programa en el que se debe tomar una decisión. Normalmente, se les denomina como “estructuras de decisión”.

ESTRUCTURAS DE DECISIÓN: CONDICIÓN, VALOR, ACCIÓN

- Cada estructura de decisión tiene una condición:
  - “¿Se ha agotado la batería del móvil?”
- Al evaluar la condición, obtenemos un valor booleano (True/False)
  - “Sí, se ha agotado” -> True; “No, aún tiene batería” -> False
- Dependiendo del valor de la condición, la estructura de decisión toma una decisión y se ejecuta la acción correspondiente:
  - IF True THEN “pon el móvil a cargar”

ESTRUCTURAS DE DECISIÓN EN PYTHON

- La sintaxis utilizada en Python para las estructuras de decisión es:

```
if CONDITION:
    ACTION
else:
    ANOTHER ACTION
```

ESTRUCTURAS DE DECISIÓN EN PYTHON (2)

- Las estructuras de decisión en Python son introducidas con “if”
- A la palabra reservada “if” le sigue la condición que se va a evaluar
- Esta condición debe ser evaluada antes de tomar una u otra acción
- El resultado de “comprobar” la condición ha de ser True o False
- Después de la condición escribiremos dos puntos “:”para hacerle saber a Python dónde termina la condición.

DECISION STRUCTURES IN PYTHON (3)

- Cuando la condición que le sigue a la palabra “if” es True, Python ejecutará las siguientes líneas que estén indentadas. La indentación puede ser una tabulación. Este cambio de sangría le permite a Python saber en qué contexto se encuentra.
- El código de la ACCIÓN sólo se ejecutará si la condición es TRUE.
- La palabra especial “else” es opcional. Sirve para indicar la acción a tomar si la condición es False.
- Si la estructura de decisión no tiene sentencia “else” y la condición no se cumple, el programa no tomará ninguna acción.

NESTING OF DECISION STRUCTURES

- Las estructuras de decisión en Python pueden ser anidadas. En otras palabras, podemos añadir una o varias estructuras de decisión dentro de otra ya existente.
- Por ejemplo:

```
If edad < 25:
    if sexo = “mujer”:
        print(“mujer joven”)
    else:
        print(“hombre joven”)
```

WHAT CAN BE INSIDE A CONDITION

- Hasta ahora sabemos que el “resultado” de evaluar una condición debe ser True o False
- Pero, ¿Cómo son estas condiciones programáticamente?
  - Booleana: `if True:` (o) `if False:` (Estas condiciones no tienen sentido pero funcionan)
  - Igual a (==): `if 5 == 4:` (o) `if age == 25:`
  - Distinto a (!=): `if 5 != 4:` (o) `if age != 25:`
  - Mayor que (>): `if 5 > 4:` (o) `if age > 25:`
  - Mayor o igual que : `if 5 >= 4:` (o) `if age >= 25:`
  - Menor que (<): `if 5 < 4:` (o) `if age < 25:`
  - Menor o igual que: `if 5 <= 4:` (o) `if age <= 25:`

¿QUÉ PUEDE HABER DENTRO DE UNA CONDICIÓN?

- Comparaciones numéricas como las anteriores
- Comparaciones de Strings o listas
- Funciones o métodos que devuelvan un valor booleano (más adelante estudiaremos lo que son las funciones)

BUCLES:
EL BUCLE WHILE

BUCLES

- A veces en un programa, es necesario ejecutar la misma acción varias veces seguidas.
- Por ejemplo, cuando nos comemos un bocadillo, vamos dando bocados hasta que [CONDICIÓN] ya no queda bocadillo O estamos llenos.
- Habría dos forma de escribir este algoritmo:
  - Repetir X veces las siguientes líneas de código: dar un bocado, comprobar... hasta que se cumpla la condición
  - Utilizar una estructura de control “loop” (bucle)
- Hay varias formas de escribir un bucle en Python pero empezaremos con WHILE

EL BUCLE WHILE

- El bucle while en Python es similar a la Estructura de Decisión en el sentido de que también evalúa condiciones. La sintaxis del bucle while en Python es la siguiente:

```
sentencia_1
while CONDICIÓN:
    sentencia_2
sentencia_3
```

> En general, la sentencia 2 hará que el valor de la condición cambie pasadas X repeticiones

De nuevo, volvemos a utilizar los dos puntos y la indentación para indicar el final de la condición y el cambio de contexto.

EL BUCLE WHILE (2)

- El funcionamiento del bucle while es muy similar al de la estructura de decisión.
- La mayor diferencia entre ambos es que la estructura de decisión ejecuta las instrucciones (o acciones) una única vez y el bucle while sigue ejecutándola mientras la condición siga devolviendo True.
- Normalmente, las instrucciones dentro del bucle while modificarán parte de la condición del bucle while.
    - Por ejemplo, en el caso del bocadillo, la instrucción será dar un bocado y la condición será si queda algo de bocadillo. Siempre que quede algo de bocadillo (condición True), le daremos un bocado y el bucle volverá a comprobar si queda bocadillo.

FUNCIONES Y MÉTODOS

FUNCIONES

- Las variables nos permiten darle un nombre a los datos que vamos a utilizar de forma que los podemos reutilizar fácilmente.
- Las funciones nos permiten darle un nombre a un algoritmo para poderlo reutilizar fácilmente.
- Una función es un bloque de código, escrito dentro de un programa, al que podemos llamar / ejecutar escribiendo el nombre / etiqueta que le hemos dado.
- Hasta la fecha conocemos dos funciones nativas de Python: print() y len()

FUNCIONES: USO Y PARÁMETROS

- Para ejecutar una función, hemos de escribir el nombre de la función seguido de los argumentos de entrada entre paréntesis:

  ```
  print(“Hello World!”)
  ```

- La función print() muestra por pantalla el texto que le pasamos como entrada dentro de los paréntesis.
- “Hello World” recibe el nombre de parámetro o argumento de entrada – le estamos diciendo a la función QUÉ es lo que queremos que nos muestre. Cambiar el parámetro de entrada cambiará lo que se imprima, pero el código que se ejecutará dentro de la función será el mismo.

FUNCIONES NATIVAS Y FUNCIONES PERSONALIZADAS

- Las funciones nativas son aquellas que ya vienen predefinidas con la instalación de Python y pueden utilizarse sin programar nada extra.
- Las funciones print() y len() son ejemplos de funciones nativas.
- También podemos crear nuestras propias funciones – funciones específicas que nos sirven para crear un programa. En este caso, nosotros elegimos su nombre, los parámetros de entrada que debe recibir y las instrucciones que queremos realizar con esos parámetros.

FUNCTIONS. RETURN VALUES.

- Hay dos tipos de funciones diferentes:
  - Funciones que simplemente ejecutan código
  - Funciones que ejecutan código y devuelven un valor (salida)
- print(“Hello World”) simplemente muestra el mensaje de entrada por pantalla. No nos devuelve nada con respecto a “Hello World”.
- len(“Hello World”) recibe el parámetro “Hello World”, cuenta el número de caracteres que contiene y devuelve (return) ese número (11). Este valor lo podemos guardar en una variable (e.g.: str_len = len(“Hello World”)) e imprimirlo.

FUNCIONES.
PARÁMETROS DE ENTRADA Y SALIDA

Podemos pensar en las funciones como pequeños algoritmos:

entrada → algoritmo → salida

argumentos de entrada → código de la función → valores de salida

Las funciones pueden tener 0 o más argumentos de entrada: `printRandomPhrase()`

También pueden devolver 0 o más valores: `first, last = getFirstAndLastLetter(word)`

METODOS

- Los métodos son funciones específicas de un determinado tipo de dato: “objeto”.
- Definir clases, objetos y programación orientada a objetos no está incluido en el temario de este curso. Por lo tanto, de momento, los métodos los veremos como funciones que pertenecen a cierto tipo de datos.
- Ejemplos de métodos específicos de strings:

  texto = “Esto es un TEXTO”

  texto.lower() <- devuelve “esto es un texto”

  texto.endswith(“O”) <- devuelve True

MÉTODOS (2)

- Los métodos “pertenecen” a una cierta variable. Hacen “algo” con el contenido de la variable. Por ejemplo, .lower() transforma todos los caracteres a minúsculas.
- Accedemos a los métodos de una variable usando el operador (.) justo después del nombre de la variable.
- De forma similar a las funciones, los métodos pueden tener 0 o más parámetros.
- De forma similar a las funciones, los métodos pueden devolver 0 o más valores.

FUNCIONES Y MÉTODOS: MÓDULOS E IMPORTACIONES

Las funciones nativas de Python pueden ser utilizadas directamente en nuestro código. Los tipos de datos nativos (int, string, bool, etc) tienen sus métodos accesibles también. Cualquier otra función que no sea nativa de Python debe ser importada antes de ser usada en nuestro código. Ejemplo: el paquete `math`.

```
import math
math.sqrt(36)  <- devuelve 6
```

Aquí encontramos de nuevo el operador “.”. Cuando accedemos a una función dentro de un objeto o módulo / librería, debemos utilizar este operador.

DEFINIR NUESTRA PROPIA FUNCIÓN

Observemos el código de la siguiente función. ¿Qué recibe, qué hace y qué devuelve?

```
def personal_info(edad, género):
    if edad <25:
        if género == “femenino”:
            print(“Mujer joven!”)
        else:
            print(“Hombre joven!”)
    else:
        print (“Persona sabia!”)
```

DEFINIR NUESTRA PROPIA FUNCIÓN (2)

- def indica el comienzo de la definición de una función.

  El nombre de la función debe aparece después de def.

  Todos los parámetros de entrada de la función han de aparecer entre paréntesis. Cada parámetro es una variable.

  Los dos puntos marcan de nuevo el final de la definición de la función y sus parámetros de entrada.

  Las instrucciones dentro de la función son como los de fuera de la función. ¡Atentos a la indentación!

- return indica el valor a devolver, si existe (por defecto es None).

  Nada más se ejecuta después de la sentencia return (aunque la función contenga más código).

CONCEPTOS BÁSICOS DE PROGRAMACIÓN

- Las primeras dos clases han cubierto los conceptos básicos de programación.
- Es importante entender bien estos conceptos porque se utilizarán en las próximas lecciones.
- Si tenéis cualquier duda, no dudéis en preguntarme.
- La sesión práctica debería ayudaros a familiarizaros con lo que hemos visto hasta el momento.

¡GRACIAS!
¿PREGUNTAS?