Facultad de Ciencias de la Administración

TECNICATURA UNIVERSITARIA EN DESARROLLO WEB

PROGRAMACIÓN I

Unidad IV – Funciones y módulos

Modularización y funciones

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración

Universidad Nacional de Entre Ríos

- **Objetivos**
  - Entender cómo definir funciones.
  - Comprender las distintas técnicas para separar las responsabilidades de una aplicación.
  - Conocer la importancia de la reutilización de código.
  - Definir funciones recursivas.
- **Temas a desarrollar:**
  - Modularización. Definición. Funciones. Definición.
  - Parámetros y argumentos. Técnicas de diseño top-down y bottom-up.
  - Módulos. Concepto. Definición. Reutilización. Concepto.
  - Recursividad. Definición.

KEEP CALM AND DIVIDE & CONQUER

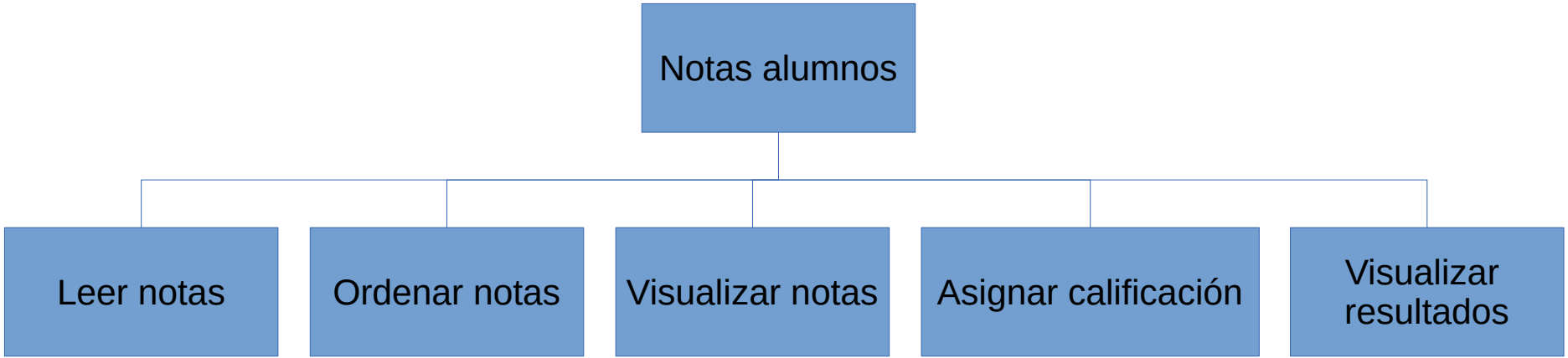
KeepCalmAndPosters.com

Modularización y diseño descendente de programas (2)

- Una de las técnicas fundamentales para resolver un problema es dividirlo en problemas más pequeños llamados **subproblemas**.
- Estos problemas pueden ser divididos repetidamente en problemas más pequeños hasta que sean solucionados.
- La técnica de dividir el problema principal en subproblemas se denomina frecuentemente, ***divide y vencerás***.
- El método de diseñar la solución de un problema principal obteniendo las soluciones de sus subproblemas se denomina ***diseño descendente (top-down design)*** debido a que se comienza en la parte superior con un problema general y se diseñan soluciones específicas a sus subproblemas.
  - Cada subproblema es deseable que sea **independiente** de los restantes y se denomina **módulo.**
  - El problema principal se resuelve con el programa principal (también llamado conductor del programa) y los subproblemas (**módulos**) mediante subprogramas.

Modularización y diseño descendente de programas (3)

- Un **subprograma** realiza una tarea concreta que se describe con una serie de instrucciones.
- La resolución de un problema comienza con una ***descomposición modular*** y luego nuevas descomposiciones de cada módulo en un proceso denominado ***refinamiento sucesivo*** (stepwise).
- Ejemplo:
  - Dadas las puntuaciones de una clase, ordenar las puntuaciones (notas) en orden decreciente; a continuación visualizar la calificación alcanzada basada en la puntuación.



Funciones

- *En el contexto de programación, una **función** es un **nombre** que se le asigna a una **secuencia de sentencias** que llevan a cabo un cómputo.*

- Una **función** nos permite definir un bloque de código ***reutilizable*** que se puede ejecutar muchas veces dentro de nuestro programa.
- Las funciones son bloques de código que se pueden reutilizar simplemente ***llamando a la función***.
- Esto permite la ***reutilización*** de código simple y elegante sin volver a escribir explícitamente secciones de código (reduce el número total de líneas).
- Esto hace que el código sea más legible, facilita la depuración y limita los errores de escritura.

Llamadas a funciones

- Ya previamente vimos invocaciones a funciones cuando usamos la función **`type(42)`**.
  - **» type(42) → <class 'int'>**
- El nombre de la **función** es **`type.`** La expresión entre paréntesis es llamada el **argumento** de la función. El **resultado**, para esta función es el tipo del argumento.
- Es común decir que la función ***"toma"*** un **argumento** y ***"retorna"*** un **resultado**. El resultado también es llamado ***valor de retorno***.
- **Python** nos provee de funciones que pueden convertir valores de un tipo en otro. La función **`int`** toma cualquier valor y lo convierte en un entero siempre que pueda, si no lo puede hacer entonces arrojará un **`ValueError.`**
  - **» int('32') → 32**
  - **» int('Hello') → ValueError: invalid literal for int(): Hello**
- **`int`** convierte valores de punto flotante en enteros, pero no puede redondearlos, simplemente trunca la parte decimal. Entonces:
  - **» int(3.9999) → 3**
  - **» int(-2.3) → -2**
- Finalmente, **`str`** convierte su argumento en un string.
- **» str(32) → '32'**
- **» str(3.15159) → '3.14159'**

Funciones matemáticas

- **Python** tiene un **módulo** que nos provee de las funciones matemáticas más comunes.
- Un **módulo** es un archivo que contiene una colección de funciones relacionadas. Antes de que podamos usar las funciones de un **módulo**, tenemos que ***importarlo*** con la sentencia **`import`**.
  - **» `import math`**
- Esta sentencia crea un objeto de tipo **módulo** que se llama **`math.`** Si se muestra, se puede obtener información sobre el mismo.
  - **» `math`** → `<module 'math' (built-in)>`
- El objeto módulo contiene las funciones y variables definidas en el módulo. Para acceder a una de las funciones, se tiene que especificar el nombre del módulo y el nombre de la función, separados por el símbolo punto **'.'.**
  - **» `ratio = signal_power / noise_power`**
  - **» `decibels = 10 * math.log10(ratio)`**
  - **» `radians = 0.7`**
  - **» `height = math.sin(radians)`**
- En el ejemplo usamos **`math.log10`** para computar la relación señal/ruido en decibeles (asumiendo que **`signal_power`** y **`noise_power`** son variables definidas) . El módulo **`math`** también provee de la función **`log`**, que computa logaritmos en base e.

Composición

- Una de las características más importantes de los **lenguajes de programación** es su habilidad de tomar pequeños elementos del programa como **variables**, **expresiones** y **sentencias** y **combinarlos**.
- Por ejemplo, el **argumento** de una función puede ser cualquier tipo de **expresión** incluso las que incluyan operadores aritméticos:
  - `x = math.sin(degrees / 360.0 * 2 * math.pi)`
- Podemos incluir también llamadas a otras funciones:
  - `x = math.exp(math.log(x+1))`
- Casi en cualquier lugar que podemos poner un valor, se puede escribir una expresión. La única excepción es que si tenemos una sentencia de asignación solamente a la izquierda puede haber un nombre de variable. Caso contrario tendremos un error de sintaxis.
  - `» minutes = hours * 60 # correcto`
  - `» hours * 60 = minutes # incorrecto! → SyntaxError: can't assign to operator`

Creando nuestras propias funciones

- **Python** nos permite crear nuestras propias funciones. Una **definición de función** especifica el ***nombre*** para la nueva función y una ***secuencia de sentencias*** que se van a ejecutar cuando la función sea invocada. Ejemplo:

```
def imprimir_letra():
    print("Hey Jude, don't make it bad.")
    print("Take a sad song and make it better.")
```

- La palabra reservada que indica que comienza la definición de una función es **`def`**. El nombre de la función es **`imprimir_letra`**.
- Las **reglas para los nombres** de función son los ***mismas*** para los **nombres de variables**.
- Los **paréntesis vacíos** luego del nombre de la función indican que no recibe **ningún argumento**.
- La primer línea de la definición de la función se llama **cabecera**; el resto **cuerpo**. La **cabecera** tiene que finalizar con el carácter ":" (dos puntos) y el cuerpo de la función tiene que estar identado. El cuerpo puede contener cualquier número de sentencias.
- La sintaxis para invocar funciones creadas por nosotros es la misma que para las funciones integradas.
  - **`imprimir_letra()`**

Creando nuestras propias funciones (2)

- Una vez que hemos definido una función podemos usarla dentro de otras funciones.
- Por ejemplo, para repetir la letra varias veces podemos escribir una función llamada **repetir_letra**:

```
def repetir_letra():
    imprimir_letra()
    imprimir_letra()
```

- Y luego llamar a la función **repetir_letra**:

```
» repetir_letra()
Hey Jude, don't make it bad.
Take a sad song and make it better.
Hey Jude, don't make it bad.
Take a sad song and make it better.
```

Definiciones y usos

- Si juntamos los fragmentos de código vistos anteriormente el programa entero se vería así:

```
def imprimir_letra():
    print(“Hey Jude, don't make it bad.”)
    print(“Take a sad song and make it better.”)
def repetir_letra():
    imprimir_letra()
    impimir_letra()
repetir_letra()
```

- La **definición de las funciones** se ejecuta tal cual el resto de las sentencias, **no generan ninguna salida** y el efecto que tienen es ***crear un objeto función***.
- Las **sentencias** dentro de la función no se ejecutan hasta que la función es **llamada**.
- La **función** debe crearse ***antes*** de ser **invocada**.

Bibliografía

- Óscar Ramírez Jiménez: ***"Python a fondo"*** 1era Edición. Ed. Marcombo S.L.. 2021.
- Allen Downey. ***"Think Python"***. 2Da Edición. Green Tea Press. 2015.
- Bill Lubanovic. ***"Introducing Python"***. 2Da Edición. O' Reilly. 2020.
- Eirc Matthes: ***"Python Crash Course"***. 1era Edición. Ed. No Starch Press. 2016.
- Zed A. Shaw: ***"Learn Python 3 the Hard Way"***. 1era Edición. Ed. Addison-Wesley. 2017.