CLASE N°17

Manejando problemas complejos

INTRODUCCIÓN

- Abstracción
  - Al enfrentar un **problema complejo**
    - **No es buena idea** sentarse frente al computador y tratar de escribir un programa que lo solucione
    - La **abstracción** es la herramienta fundamental para **enfrentarlos**
  - Aplicamos abstracción con tres objetivos:
    - Para entender el problema: **abstracción de problemas**
    - Para modelar los datos involucrados: **abstracción de datos**
    - Para modelar el proceso que soluciona el problema: **abstracción de procesos**

ABSTRACCIÓN DE PROBLEMAS

- Los problema del mundo real se presenta con **enunciados**
  - Que son **imperfectos**
  - Debemos obtener un **planteamiento** más concreto **(modelo del problema)**
    - Identificando **entidades** y **aspectos importantes**
    - **Dando nombres** a estos elementos
    - Definiendo qué **operaciones**, conocidas o no, pueden aplicarse
    - Completando **información omitida**
    - Ignorando **datos irrelevantes**

ABSTRACCIÓN DE DATOS

- El manejo de los datos puede **complejizar** el problema
  - Es mejor **separar esta complejidad**
    - Definiendo un **modelo del tipo de dato requerido**
    - Indicando **qué propiedades y funciones** estarían definidas
    - Pero sin establecer **una estructura interna** específica
  - Al implementar la solución, se debe **implementar** el modelo de datos
    - Algunas de las funciones pueden ser **bastante complejas**
    - Que pueden originar **subproblemas**

ABSTRACCIÓN DE PROCESOS

- Un problema planteado define un **objetivo**
  - Debemos pensar **una secuencia de pasos** que permite alcanzarlo
    - **Finita**, con un **orden lógico**
- Normalmente esto nos lleva a un **modelo inicial de la solución (estrategia)**
  - Que requiere **refinamiento**
  - Cada paso se convierte en un **subproblema**, asociado a un **sub-objetivo**
  - Cuya solución involucra **secuencias de pasos más específicas**

REFINAMIENTO ALGORÍTMICO

- El modelo de la solución toma la **forma de programa**
  - **Pasos manejables** se expresan como **sentencias**
  - **Pasos complejos** se expresan como la **llamadas a subrutinas**
  - Cada subrutina **se refina** de igual manera, en forma **independiente**
  - Hasta que todos los pasos pueden ser **implementados “fácilmente”** en un lenguaje de programación

IMPLEMENTACIÓN

- La **implementación** es la **operación inversa** a la abstracción
  - Los **detalles** se completan a **conveniencia** del lenguaje de programación usado
  - Pero debe **seguirse** el **modelo de solución** propuesto
    - El **bloque principal** es la estrategia general obtenida
    - **Cada función** debe construirse **separadamente**, con la mayor **encapsulación** posible
      - **entradas = parámetros**, **salidas = valores de retorno**
    - Debe decidirse una **representación interna** para el modelo de datos y **construir las funciones** que tiene definidas

OBJETOS EN PYTHON

- Python dispone **nativamente** de varias estructuras de datos complejas en la forma de **objetos**
  - Una **instancia** en la memoria del computador de una **clase**
  - **No conocemos** cómo guardan su contenido
  - **Interactuamos** con el objeto y su contenido a través de **métodos**
- Conocemos varias **clases** de objetos:
  - **Inmutables**: Strings, tuplas y archivos
  - **Mutables**: Listas, filas, pilas, conjuntos, diccionarios

STRINGS

- Clase `str`
  - Secuencias de inmutable de **caracteres**
    - Para crear objetos `str` usamos **texto constante en comillas** o la función nativa `str(<valor numérico>)`
    - Métodos permiten obtener **nuevos strings** con el contenido original modificado (por ejemplo, en mayúsculas o sub-strings)
    - El método `.split(<patrón>)` permite dividir el string y obtener una lista de palabras
    - Podemos **iterar** secuencialmente sobre los caracteres o accederlos mediante **indexación posicional**

```
for char in <string>:
    <uso del caracter char>
```

```
for i in range(0, len(<string>):
    <uso del caracter <string>[i]>
```

ARCHIVOS

- Clase `file`
  - Colección de **líneas de texto**
    - Para crear objetos `file` usamos la función nativa **`open()`**
    - Para **finalizar** objetos `file` usamos el método **`.close()`**
    - Para **escribir un string** en un archivo, abierto en modo escritura, usamos el método **`.write()`**
    - Para **leer una línea** del archivo, abierto en modo lectura, usamos el método **`.readline()`**
    - Podemos **iterar** (secuencialmente) sobre las líneas de texto:

```
for linea in <objeto archivo abierto>:
    <sentencias que usan el contenido de linea>
```

TUPLAS

- Clase `tuple`
  - Secuencias de inmutable de **valores**
    - Para crear objetos `tuple` usamos valores constantes separados por comas o la función nativa `tuple(<valor iterable>)`

```
par = 2, 5    parVacio = tuple()    unoSolo = "Hola",
```

    - Inmutable significa que **no podemos agregar o quitar** elementos (pero elementos mutables pueden cambiar sin problemas)
    - Podemos **iterar** secuencialmente sobre los valores o accederlos mediante **indexación posicional**

```
for elem in <tupla>:
    <uso del valor elem>
```

```
for i in range(0, len(<tupla>):
    <uso del valor <tupla>[i]>
```

LISTAS

- Clase `list`
  - Secuencias de mutable de **valores**
    - Para crear objetos `list` usamos valores constantes separados por comas entre paréntesis cuadrados o la función nativa `list(<valor iterable>)`
    - Usamos los métodos `.insert()` o `.append()` para agregar valores
    - Podemos **iterar** secuencialmente sobre los valores o accederlos mediante **indexación posicional**

```
for elem in <lista>:
    <uso del valor elem>
```

```
for i in range(0, len(<lista>):
    <uso del valor <tupla>[i]>
```

    - Usamos el operador `del` para eliminar valores

```
del <lista>[i]
```

    - ¡Cuidado con ir **iterando y eliminando** valores!

FILAS Y PILAS

- Clase `list`
  - Una lista se comporta como **fila** si:
    - Agregamos valores usando el método `.append()`
    - Quitamos valores usando el método `.pop(0)`
  - Una lista se comporta como **pila** si:
    - Agregamos valores usando el método `.append()`
    - Quitamos valores usando el método `.pop()`

CONJUNTOS

- Clase `set`
  - Colección de **valores inmutables**
    - Para crear objetos `set` usamos la función nativa `set(<valor iterable>)`
    - Para **agregar un elemento** usamos el método `.add()`; se ignoran elementos repetidos
    - Para **quitar un elemento** usamos el método `.discard()`; eliminar elementos que no existen no tiene efecto
    - Podemos **iterar** (secuencialmente) sobre los elementos:

```
for elem in <conjunto>:
    <uso del elemento elem>
```

    - Existen **métodos** para **operar conjuntos**

DICCIONARIOS

- Clase `dict`
  - Colección de **valores indexados con llaves inmutables**
    - Para crear objetos `dict` usamos la función nativa `dict()`
    - Para **agregar un elemento** usamos indexación; llaves no pueden repetirse (valores se sobre escriben)

```
<diccionario>[<valor llave>] = valor
```

    - Para **quitar un elemento** usamos el operador `del`; no podemos eliminar elementos con llaves inexistentes

```
if <diccionario>.has_key(<valor llave>):
   del <diccionario>[<valor llave>]
```

    - Podemos **iterar** (secuencialmente) sobre las llaves:

```
for llave in <diccionario>.keys():
   <uso del elemento <diccionario>[<llave>]>
```

CONSULTAS

¿CONSULTAS?