Introducción a la Programación

Prof. Agustín Gravano

Primer semestre de 2022

Clase teórica 1: Introducción a la materia

Introducción a la Programación

Esta materia ofrece un recorrido introductorio y abarcativo por los principales temas de la programación imperativa.

Se presentan elementos básicos como las variables, los tipos de datos, la memoria, el estado de un programa, las estructuras de control, las funciones y la recursión.

Además se introducen conceptos fundamentales como:

- la especificación de funciones (para describir la tarea a realizar),
- la complejidad algorítmica (para estimar el tiempo de ejecución),
- la verificación de programas (para intentar asegurar que un programa funcione correctamente).

Nuestros objetivos son aprender a pensar algorítmicamente e incorporar buenas prácticas de programación.

¡Saber programar no es solo saber escribir código!

Lenguaje de programación

Pregunta frecuente: ¿Qué lenguaje de programación vamos a usar?

Respuesta corta: Python (versión 3 o superior)

Respuesta larga: No importa demasiado. Los conocimientos básicos de programación son comunes a la gran mayoría de los lenguajes.

Analogía con los lenguajes naturales: En la niñez, aprendemos a interactuar socialmente: saludar, agradecer, pedir cosas, etc. Esas acciones son independientes del idioma. Al aprender un idioma nuevo, solo necesitamos que nos expliquen cómo ejecutarlas.

Objetivo de esta materia: Incorporar elementos básicos de programación, que son independientes del lenguaje usado (Python, R, C++, Java, etc.). Al aprender un lenguaje nuevo, no deberían necesitar que les expliquen esos conceptos.

¿Qué vamos a haber aprendido al terminar la materia?

- Machine learning, big data, business analytics... Todavía no. :-)
  De eso (y mucho más) se trata **la carrera**; en esta materia daremos los primeros (importantes) pasos en esa dirección.
- Resolver problemas de cierta dificultad, con control y confianza en nuestro código.
- Procesar archivos con datos (punto de partida para todo lo demás).
- Ejemplo: leer bases de datos disponibles públicamente, modelar bien los datos, responder preguntas más o menos complejas.

Temas de la materia (en orden más o menos cronológico)

1. Problema, especificación, algoritmo y programa. Lenguaje Python.
2. Tipos de datos, expresiones, variables, memoria, estado del programa.
3. Funciones, especificación de funciones. Concepto de encapsulamiento.
4. Tipos de errores de los programas, testing de unidad.
5. Control de flujo: condicionales (`if`) y ciclos (`while`).
6. Listas: secuencias de elementos. Tipos de datos compuestos.
7. Lectura/escritura de archivos.
8. Representación de la información.

---

9. Complejidad algorítmica, tiempo de ejecución de un programa.
10. Problemas clásicos: ordenamiento y búsqueda.
11. Más tipos de datos compuestos: tuplas, conjuntos y diccionarios.
12. Definición de tipos propios: clases y objetos.
13. Recursión algorítmica.

Bibliografía

Allen B. Downey, Jeffrey Elkner, Chris Meyers,
"Aprenda a Pensar como un Programador (con Python)"
Segunda edición, Green Tea Press, 2002

- Libro abierto, disponible para bajar gratis en forma legal.
- Trabaja con Python 2 (ya deprecado), pero con algunos cuidados se puede usar sin problemas.
- Está en español.
- Hay una 3ra edición, disponible en inglés, que usa Python 3.
- Tener en cuenta las diferencias entre Python 2 y 3. Las dos más relevantes para esta materia son las dos primeras: `print` y división de números enteros.

Plantel docente de la materia

Profesor a cargo:

Agustín Gravano

agravano@utdt.edu

Docentes auxiliares:

Augusto
González Omahen

Camila
Di Ielsi

Nahuel
Díaz

Organización de la materia

- **Lunes y miércoles:** clases teórico-prácticas, a cargo de Agustín.
- **Viernes:** clases prácticas, de ejercitación, a cargo de un/a auxiliar.
- **Horarios de consulta:** A definir.

Campus Virtual:

- Materiales (slides de las clases, guías de ejercicios, enunciados de trabajos prácticos, ejemplos, datos, etc.).
- Foros para consultas.
- Leer con cuidado la modalidad de evaluación de la materia.
- Leer con cuidado las reglas disciplinarias.

¿Qué se espera de cada estudiante?

Durante la clase:

- Tomar apuntes. Las slides se publican con los contenidos mínimos, pero hacemos muchos comentarios y anotaciones.
  Consejo: usar un cuaderno por materia para apuntes, ejercicios, etc.
- Resolver los ejercicios. Las explicaciones se alternan con ejercicios para hacer en el momento.
- Participar: Hacer preguntas (¡es normal no entender algo!) y responder las preguntas de los docentes.

¿Qué se espera de cada estudiante?

Después de la clase:

- Estar al día con los temas de la materia. No permitir que se forme una bola de nieve.
- Ejercitar. Resolver las guías de ejercicios. Al final de cada clase indicamos hasta dónde pueden avanzar.
- Leer el libro y los apuntes, que **complementan** lo visto en clase.
- Estudiar en grupo. Formar un grupo de estudio es una excelente idea para resolver ejercicios juntos y aclararse dudas.
- Usar el foro del campus virtual para consultarnos por dudas con ejercicios u otros temas de la materia.
  - ¡Leer antes de consultar! Quizá otros ya preguntaron lo mismo.
  - Por favor, **no** contactarnos por mail u otro medio.
- Cuidado con la web. Hay mucho material online, muchas veces de mala calidad. En general busca resolver cuestiones puntuales en un lenguaje; no busca aprender a pensar algorítmicamente y adquirir buenas prácticas de programación (nuestros objetivos).

¿Preguntas?

Programar (bien) es mucho más que escribir código.

Hagamos un recorrido rápido por el ciclo de desarrollo de un programa:

Especificación $\rightarrow$ Algoritmo $\rightarrow$ Programa $\rightarrow$ Verificación

Ejemplo: ¿cuántas veces aparece la letra `'t'` en un texto?

Programar (bien) es mucho más que escribir código

Ejemplo: ¿cuántas veces aparece la letra `'t'` en un texto?

Primero, escribimos una especificación de un programa que resuelva el problema: una descripción (tan clara como sea posible) de **qué** debe hacer ese programa.

- Entrada: Un texto (una cadena de caracteres) de longitud arbitraria, que llamamos `txt`.
- Salida: Un número entero ≥ 0, igual a la cantidad de apariciones de `'t'` en `txt`, ya sea en minúscula o mayúscula.

Además construimos ejemplos que describan el comportamiento esperado.

Entrada	Salida
`txt='pato'`	1
`txt='tecnología digital'`	2
`txt='Universidad Torcuato Di Tella'`	3
`txt='xyz'`	0
`txt='ttTT'`	4
...	...

Programar (bien) es mucho más que escribir código

Segundo, pensamos un algoritmo: una descripción abstracta de **cómo** resolver el problema, escrita en español o en pseudocódigo, para seres humanos (no para computadoras).

1. Pongo un contador en 0.
2. Recorro los caracteres de `txt`.
3. Para cada carácter, si es igual a `'t'` o `'T'`, sumo 1 al contador.
4. Cuando llego al final de `txt`, devuelvo el valor del contador.

Normalmente, hay muchísimos algoritmos posibles para cada problema.

Programar (bien) es mucho más que escribir código

Tercero, traducimos nuestro algoritmo a algún lenguaje de programación. Lo convertimos en un programa, para ser ejecutado en una computadora.

(Elegimos usar Python, pero si hubiéramos elegido otro lenguaje, sería todo muy parecido.)

```
1  def contar_t(txt:str) -> int:
2      contador:int = 0
3      for caracter in txt:
4          if caracter=='t' or caracter=='T':
5              contador = contador + 1
6      return contador
```

Recién en esta etapa resolvemos las cuestiones de implementación específicas del lenguaje.

Programar (bien) es mucho más que escribir código

Normalmente, hay muchísimos programas posibles para cada algoritmo.

```python
def contar_t(txt:str) -> int:
    contador:int = 0
    i:int = 0
    while i<len(txt):
        if txt[i]=='t' or txt[i]=='T':
            contador = contador + 1
        i = i + 1
    return contador
```

```python
def contar_t(txt:str) -> int:
    contador:int = 0
    if txt != '':
        contador = contar_t(txt[1:])
        if txt[0]=='t' or txt[0]=='T':
            contador = contador + 1
    return contador
```

En el **paradigma de programación imperativa**, un programa es una secuencia finita de instrucciones: operaciones que transforman datos (el *estado* del programa), o bien modifican el flujo de ejecución.

En la materia vamos a ver algunos elementos de los paradigmas **funcional** y **orientado a objetos**.

Programar (bien) es mucho más que escribir código

Por último, realizamos una verificación del programa. Queremos saber: ¿nuestro programa hace lo esperado? Para ello, vamos a hacer testing de unidad con los ejemplos que habíamos construido.

```
1  import unittest
2  class TestBuscar(unittest.TestCase):
3      def test_contar_t(self):
4          self.assertEqual(contar_t('pato'), 1)
5          self.assertEqual(contar_t('tecnología digital'), 2)
6          texto = 'Universidad Torcuato Di Tella'
7          self.assertEqual(contar_t(texto), 3)
8          self.assertEqual(contar_t('xyz'), 0)
9          self.assertEqual(contar_t('ttTT'), 4)
10 unittest.main()
```

El objetivo de la verificación es ganar confianza de que el programa es correcto, aunque nunca vamos a tener plena seguridad de ello. :-(

¡Pero vayamos de a poco!

Eso fue un primer pantallazo a los temas que vamos a ir cubriendo en las próximas semanas.

Por ahora, tengan presente el ciclo de desarrollo de un programa, que nos va a acompañar durante toda la ~~materia~~ carrera (profesional):

$$\text{Especificación} \rightarrow \text{Algoritmo} \rightarrow \text{Programa} \rightarrow \text{Verificación}$$

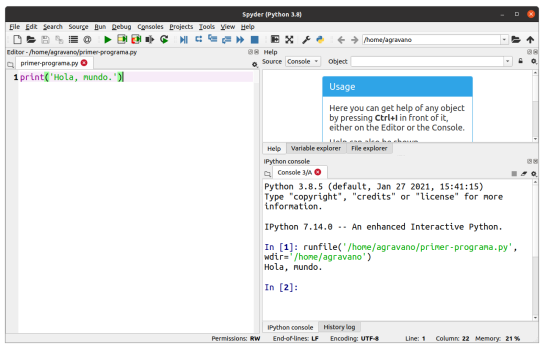
Antes de terminar, veamos un poco del entorno de programación que vamos a usar para trabajar con Python.

Entorno de programación

Recomendamos instalar el framework Anaconda con Python 3.9:
https://www.anaconda.com/products/individual#Downloads

- Ante la duda, descargar la opción de 64 bits.
- Durante la instalación, si aparece la opción de modificar el *path*, responder que sí (esto permite ejecutar programas en Python en forma más directa).

Anaconda incluye un entorno de desarrollo integrado (IDE) llamado Spyder, que tiene editor de texto, consola `ipython` y otras herramientas para escribir, probar y depurar código en Python.



Hay otros entornos: VSCode, PyCharm, etc. Pueden usar el que prefieran.

Si tienen problemas o dudas, pidan ayuda en el Foro del campus virtual.
Mientras tanto, pueden empezar a trabajar en un intérprete online de Python:
https://www.programiz.com/python-programming/online-compiler/
Para los primeros temas de la materia está bien; después necesitaremos Spyder.

Repaso de la clase de hoy

- Introducción y organización de la materia.
- Ciclo de desarrollo de un programa:
  Especificación $\rightarrow$ Algoritmo $\rightarrow$ Programa $\rightarrow$ Verificación
- Entorno de programación Spyder.

Bibliografía complementaria:

- APPP2, capítulo 1
- HTCSP3, capítulo 1

Para la próxima clase:

- Tener funcionando un entorno con Python3 (ej: Spyder).
- Escribir y ejecutar un programa que imprima por pantalla el mensaje `'Hola, mundo.'`

Comentarios anónimos sobre la materia:

- https://tinyurl.com/td1-feedback-1s-2022
- Críticas, ideas, sugerencias, etc. son bienvenidas todo el semestre.