Inducción al curso de Estructuras de datos

Héctor Xavier Limón Riaño

April 3, 2024

Contents

1 Preguntas de repaso

- ¿Qué es una computadora?
- ¿Qué es un programa?
- ¿Cómo ejecutamos programas?
- ¿Cómo creamos programas?
- ¿Qué es un lenguaje de programación?
- ¿Qué es un compilador?
- ¿Qué es un intérprete?
  - Portabilidad:
    - Capacidad de poder ejecutar un programa en cualquier plataforma, siempre y cuando tengas el intérprete

2 Cómo ser un buen programador/desarrollador

- Aprende sobre algoritmia y desarrolla tus habilidades algorítmicas
- Estudia temas de computación
  - Sistema operativos
  - Redes
  - Ciberseguridad
  - Criptografía
  - Servidores
  - Hardware
  - Teoría de compiladores
- Conoce tus herramientas:
  - Lenguaje de programación
  - SO
  - Shell de sistema
  - Editor de código

  - Depuradores
  - Perfiladores
  - Testing
  - Multi-procesamiento
  - Virtualización y contenedores
  - Mecanismos de comunicación interproceso
  - Manejadores de bases de datos
- Aprende varios paradigmas de programación
  - Orientado a objetos
  - Procedural
  - Funcional
  - Lógico
- Aprende sobre Ingeniería de Software
  - Análisis de requisitos
  - Diseño
  - Construcción
  - Prueba
  - Despliegue

3 Tipos de software

3.1 Por plataforma de ejecución

- Escritorio
- Web
- Móvil
- IoT
- Red

3.2 Por propósito:

- Aplicación
  - Para usuarios finales
  - Aplicaciones de escritorio, web, móviles
  - Interés especial en requisitos de usuario
- Sistema
  - Software de base, como SO, firmwares (IoT), compiladores, intérpretes, ligadores, etc
  - Usualmente de propósito más general
  - Conocimiento de hardware y bajo nivel
- Automatización
  - Facilitar tareas de administración de recursos: redes, servidores
  - Muy asociado al scripting
  - Muchas veces son de uso personal o para un equipo pequeño
  - Necesario en tareas de ciberseguridad como el pentesting
  - Se requiere habilidad para resolver problemas rápidamente
- Utilería
  - Pensado para que lo usen otros tipos de software
  - Algoritmos y protocolos criptográficos, servicios de SO (acceso a sistema de archivos, procesos, etc), APIs web, etc.

3.3 Por tipo de interfaz de usuario

- Línea de comandos
  - De texto
  - Orientadas a teclado
  - No requieren de una interfaz gráfica, sólo de la consola de texto
  - Requiere de un shell de sistema (Bash, Zsh, Fish, Powershell, etc.)
  - Posibilidad de manipular entrada y salida estándar
  - Composición de comandos

  - El software de automatización suele ser de línea de comandos

- TUIs (Text User Interface)
  - De texto
  - Orientadas a teclado
  - No requieren de una interfaz gráfica, sólo de la consola de texto
  - Interacción basada en combinaciones de teclado

- GUIs (Graphical User Interface)
  - Interfaces gráficas con widgets (botones, áreas de texto, menús, etc.)
  - Lo más común para sistemas de escritorio, web y móviles
  - Orientadas a uso de mouse o pantallas táctiles

4 Para este curso

- Escritorio
- Orientación a software para automatización
- Interfaz de línea de comandos

4.1 Por lo tanto se requieren estas herramientas

- Lenguaje de scripting junto con su intérprete
- Emulador de terminal y shell de sistema
- Editor de texto (con soporte del lenguaje)

4.2 En particular

- Python como lenguaje de programación
- Cualquier SO
- Cualquier emulador de terminal y shell de sistema
- Cualquier editor de texto con soporte para Python. Algunas opciones:
  - VScode

  - Pycharm
  - Sublime
  - Vim
  - NeoVim
  - Emacs

5 Python

5.1 Generalidades

- Lenguaje de programación multi-paradigma
- Interpretado
- Open source
- Gran comunidad y paquetes de terceros
- En el top 3 de la mayoría de listas sobre lenguajes
- Probablemente el lenguaje más utilizado en Ciberseguridad y automatización de infraestructura
- Muy utilizado en otros ámbitos como desarrollo web, ciencias de datos e IA en general
- Alta demanda en puestos de trabajo

5.2 Cosas más técnicas importantes

- Estilo de programación de scripting
- Comportamiento dinámico
- Estilo de desarrollo REPL (Read-Eval-Print-Loop)
- Se puede trabajar directamente en el intérprete o un archivo a parte (extensión .py)
- Para ejecutar un script Python desde la línea de comandos
- El comando python sin argumentos permite entrar al intérprete

```
python script.py # ejecutar script
python # entrar al intérprete
```

5.3 Hola mundo

```
print('hola mundo')
```

5.4 Tarea

- Traer ambiente de trabajo preparado
- Demostrar la ejecución del clásico hola mundo

5.5 Crash course

5.5.1 Cosas generales

- No hay método main ni clase principal
- Sólo hay comentarios de línea, empezando con `#`
- Hay muchas diferencias sintácticas con respecto a lenguajes como C y derivados (C++, Java, C#, JavaScript, etc.)
- También hay muchas similitudes:
  - Estilo imperativo
  - Mismas sentencias de control principales (`if`, `for` y `while`)
  - Declaración similar de variables
  - Casi los mismos operadores
    - Excepto algunos operadores lógicos como `and`, `or` y `not`
    - No hay operadores tipo `++`
  - Manipulación similar de expresiones
  - No es necesario terminar las sentencias con `;`

5.5.2 Identación

- En Python la identación es un elemento sintáctico para establecer el nivel de los bloques, ten cuidado
- Un buen editor te ayuda con la identación al usar la tecla Tab, sin insertar realmente el carácter de tabulación

5.5.3 Expresiones

- Los dos elementos sintácticos principales de un lenguaje son las expresiones y las sentencias
- Aprender un lenguaje de programación en gran parte se trata de aprender estos dos conceptos en el contexto del lenguaje
- Una expresión es cualquier cosa que tiene un valor asociado:
  - Valor literal (como `4`, `"hola"`, `True`, etc.)
  - Variables
  - Expresión matemática (`3 + 4`)
  - Expresión lógica, también llamada booleana (`4 < 3`)
  - Invocación de una función (el valor es lo que regresa la función)
  - Operadores especiales como punto, corchetes, llaves, entre otros
- Existen operadores que conectan expresiones para formar expresiones más complejas
- El operador especial `()` cambia el orden de evaluación de expresiones, ante la duda de la precedencia úsalo

5.5.4 Declaración de variables

- La declaración es un tipo de sentencia, no una expresión, no tiene valor asociado
- No es necesario establecer el tipo, aunque si lo hay (más sobre esto después)
- Una variable puede cambiar de tipo (a diferencia de C)
- Se utiliza el operador = que se lee como "toma el valor", no es un "igual a" para eso está el operador lógico ==
- Un identificador es cualquier cosa que nombra el programador
- Un identificador no pude empezar con un número, ni tampoco puede incluir ciertos caracteres

```
#Forma:
identificador = expresion

# Ejemplos
x = 5
x = 5 + 5
x = 5 + 5 * 5 # 30, primero se evalúa 5 * 5
x = (5 + 5) * 5 # 50, primero se evalúa 5 + 5
x = suma(5, 5) # invocación a función
x = suma(5, 5) - 1 # válido porque ambas son expresiones enteras
x = 'hola' # cadena, se puede cambiar el tipo
x = 'hola ' + 'mundo' # válido porque + se refiere a concatenación
x = 'hola' + 4 # error, incompatibilidad de tipos
y = 4
x += y # equivalente a x = x + y, hay variantes -=, *=, /=, %=
x = 4**2 # 4 elevado al cuadrado
x = 4 % 2 # operación de módulo
t = True or False
t = not t # intercambiar valor de verdad
b = 4 < 5 # b es True
b = 4 != 4 # b es False
```

5.5.5 Listas

- Es probablemente la estructura de datos más importante en Python
- Más adelante en el curso se ve a fondo
- Pero vale la pena ver su cómo se usa de manera básica para dar ejemplos
- Muy parecido a arreglos en otros lenguajes, es una estructura lineal indexada
- Los elementos pueden ser de cualquier tipo
- Los índices pueden ser negativos

```
lista = [1, 2, 3]
x = len(lista) # longitud de la lista
y = x[0] # primer elemento
y = x[2] # último elemento
```

```
y = x[-1] # tambíen último elemento
x[1] = 3 +3 # se pueden asignar valores nuevos
otra = ['hola', 4.4, True] # se pueden mezclar tipos
```

5.5.6 Sentencias de control

- No tienen un valor asociado, no son expresiones
- Sirven para cambiar el flujo de control del programa, que por defecto es de arriba abajo y de izquierda a derecha

1. If
    - Crea bifurcaciones en el flujo de control, de acuerdo a condiciones referentes al estado del programa

```
# Forma
if expresion_booleana:
    sentencias_y_expresiones
elif otra_expresion_booleana: # elif es opcional, puede haber varios
    sentencias_y_expresiones
else: # opcional, debe ir al final
    sentencias_y_expresiones

# Ejemplos
if True:
    print('entra al if')

entra al if



if 3 < 2 or True:
    print('entra a if')
elif 6 == 6.0:
        print('entra al primer elif')
elif 5 > 3:
    print('entra al segundo elif')
else:
    print('entra al else')
```

```
entra a if

l = [1, 3, 5, 6]
if 5 in l:
    print('está el 5')
else:
    print('no está el elemento')

está el 5
```

2. While

   - Para crear bucles en el flujo de control
   - Permite repetir instrucciones de acuerdo a una condición lógica
   - El cuerpo de la sentencia debería provocar que eventualmente la condición sea falsa (a menos que se quiera un ciclo infinito)
   - Es el tipo más general de bucle (pude comportarse igual que un for)

```
  # Forma
while expresion_booleana: # se itera mientras True
    sentencias_y_expresiones
```

   - Dentro de bucles (tanto while como for), pueden usarse las sentencias especiales `break` y `continue`
   - `break` causa que se rompa el ciclo sin importar la condición lógica.
   - Notar que `return` (visto más adelante para terminar funciones), también rompe los ciclos, aunque también termina toda la función

```
x = 0
while True: # ciclo infinito
    print(x)
    if x > 2:
        break
    x += 1

0
1
2
3
```

- `continue` por su parte hace que el ciclo se vaya a la siguiente iteración

```
# sólo imprimir impares
x = 0
while x < 10:
    x += 1
    if x % 2 == 0: # es par
        continue
    print(x)
```

```
1
3
5
7
9
```

```
1
3
5
7
9
11
```

3. For

   - Es un tipo especial de bucle pensado para recorrer estructuras de datos lineales
   - No debería usarse para otra cosa, para otros casos está `while`
   - En general, existen dos tipos de `for` en los lenguajes de programación:
     - `Estilo C`: tres partes: sentencia a ejecutar antes de entrar al ciclo, expresión booleana que se debe cumplir para iterar y sentencia que es ejecuta al final de cada iteración
     - `for each` En cada iteración se toma el siguiente elemento de una estructura de datos
     - Python sólo cuenta con el estilo `for each`, aunque es posible simular el `estilo C`

```
        # Forma
        for elemento in estructura:
            sentencias_y_expresiones

for palabra in ['hola', 'mundo', 'mundial']:
    print(palabra)

hola
mundo
mundial
```

- Para simular `estilo C` se usa la función especial `range`
- `range` regresa una secuencia de números de acuerdo a sus argumentos

```
for caracter in 'hola':
  print(caracter)

h
o
l
a

lista = ['hola', 'mundo', 'mundial'] # tamaño 3, posiciones del 0 al 2
for i in range(len(lista)): # regresa los  números 0, 1 y 2, el 3 es no inclusivo
    print(lista[i])

hola
mundo
mundial
```

5.5.7 Funciones

- Sirven para encapsular funcionalidad
- Reciben entradas (parámetros) y producen un valor de salida
- Son la forma más básica de dividir problemas
- Dividir es la forma en que los humanos lidiamos con la complejidad

- El diseño de software trata esencialmente sobre el manejo de la complejidad en varios niveles
- Entre mejor seas dividiendo problemas, mejor programador serás
- El consejo fundamental es tratar de que cada función haga una y sólo una cosa bien
- A lo anterior se le llama `alta cohesión` y es posiblemente la propiedad de diseño más importante
- Una función puede regresar un valor, aunque también puede no hacerlo

```
# Forma general
def identificador(param1, param2, paramN):
    sentencias_y_expresiones
    return expresion # opcional

def saludar(nombre):
    print('hola ' + nombre) # no hay return

def sumar(numero1, numero2):
    return numero1 + numero2

def externa():
    def interna():
        return 1
    return 2

saludar('Pepe')
print(sumar(55, 3))
print(sumar("hola", " mundo")) # se pude pero no es semánticamente bueno
```

```
hola Pepe
58
hola mundo

hola Pepe
58
```

- Notar que no hay tipo de retorno, se resuelve de forma dinámica

- Trata de usar buenos nombres:
  - El nombre de tu función debería ser un verbo, posiblemente acompañado de un sustantivo: `registrar_usuario`, `econtrar_mayor`
  - El nombre de tus parámetros (y variables en general) deberían ser sustantivos representativos, evita usar una sola letra como nombre de variable (salvo en casos donde el nombre es bien conocido como la variable `i`)
- Considera que en gran parte, lo que programamos no es para nosotros mismos, ayúdale al que intenta entender tu código (o ayúdale a tu yo del futuro que intenta entender lo que hiciste antes)