Think Python
How to Think Like a Computer Scientist
(Traducción al español)

Allen Downey

Traducción: Brayan Calderon

Capítulo 1

El camino del programa

El objetivo de este libro es enseñarte a pensar como un **científico de la computación**. Esta forma de pensar combina algunas de las mejores características de las matemáticas, la ingeniería y las ciencias naturales. Como los matemáticos, los científicos de la computación usan lenguajes formales para denotar ideas (específicamente, computaciones). Como los ingenieros, diseñan cosas ensamblando componentes en sistemas y evaluando compromisos entre alternativas. Como los científicos, observan el comportamiento de sistemas complejos, forman hipótesis y prueban predicciones.

La habilidad más importante para un científico de la computación es la **resolución de problemas**. Resolver problemas significa tener la capacidad de formular problemas, pensar creativamente sobre soluciones y expresar una solución de forma clara y precisa. Como verás, el proceso de aprender a programar es una excelente oportunidad para practicar habilidades de resolución de problemas. Por eso, este capítulo se llama *"El camino del programa"*.

En un nivel, estarás aprendiendo a programar, una habilidad útil por sí misma. En otro nivel, estarás usando la programación como un medio para alcanzar otros fines. A medida que avancemos, ese propósito se volverá más claro.

1.1. ¿Qué es un programa?

Un programa es una secuencia de instrucciones que especifican cómo ejecutar una computación. La computación puede ser algo matemático, como solucionar un sistema de ecuaciones o determinar las raíces de un polinomio, pero también puede ser una computación simbólica, como buscar y reemplazar el texto de un documento o (aunque parezca raro) compilar un programa. Las instrucciones (comandos, órdenes) tienen una apariencia diferente en lenguajes de programación diferentes, pero existen algunas funciones básicas que se presentan en casi todo lenguaje:

Entrada: Recibir datos del teclado, o un archivo u otro aparato.

Salida: Mostrar datos en el monitor o enviar datos a un archivo u otro aparato.

Matemáticas: Ejecutar operaciones básicas de matemáticas como la adición y la multiplicación.

Operación Condicional: Probar la veracidad de alguna condición y ejecutar una secuencia de instrucciones apropiada.

Repetición: Ejecutar alguna acción repetidas veces, normalmente con alguna variación.

Lo crea o no, eso es todo. Todos los programas que existen, por complicados que sean, están formulados exclusivamente con tales instrucciones. Así, una manera de describir la programación es: El proceso de romper una tarea en tareas cada vez más pequeñas hasta que estas tareas sean suficientemente simples para ser ejecutadas con una de estas instrucciones simples. Quizás esta descripción sea un poco ambigua. No se preocupe. Lo explicaremos con más detalle con el tema de los algoritmos.

1.2. Ejecutando Python

Uno de los desafíos de empezar con Python es que podrías tener que instalar Python y software relacionado en tu computadora. Si estás familiarizado con tu sistema operativo, y especialmente si te sientes cómodo con la interfaz de línea de comandos, no tendrás problemas para instalar Python. Pero para principiantes, puede ser doloroso aprender sobre administración de sistemas y programación al mismo tiempo.

Para evitar ese problema, recomiendo que comiences ejecutando Python en un navegador. Más adelante, cuando te sientas cómodo con Python, haré sugerencias para instalar Python en tu computadora. Hay varias páginas web que puedes usar para ejecutar Python. Si ya tienes una favorita, adelante y úsala. De lo contrario, recomiendo PythonAnywhere. Proporciono instrucciones detalladas para empezar en http://tinyurl.com/thinkpython2e.

Hay dos versiones de Python, llamadas Python 2 y Python 3. Son muy similares, así que si aprendes una, es fácil cambiar a la otra. De hecho, solo hay unas pocas diferencias con las que te encontrarás como principiante. Este libro está escrito para Python 3, pero incluyo algunas notas sobre Python 2. El intérprete de Python es un programa que lee y ejecuta código Python. Dependiendo de tu entorno, podrías iniciar el intérprete haciendo clic en un icono, o escribiendo python en una línea de comandos. Cuando se inicia, deberías ver una salida como esta:

```
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more
   information.
```

Las primeras tres líneas contienen información sobre el intérprete y el sistema operativo en el que se está ejecutando, así que podría ser diferente para ti. Pero deberías verificar que el número de versión, que es 3.4.0 en este ejemplo, comience con 3, lo cual indica que estás ejecutando Python 3. Si comienza con 2, estás ejecutando (lo adivinaste) Python 2. La última línea es un prompt que indica que el intérprete está listo para que ingreses código. Si escribes una línea de código y presionas Enter, el intérprete muestra el resultado:

```
>>> 1 + 1
2
```

Ahora estás listo para comenzar. De aquí en adelante, asumo que sabes cómo iniciar el intérprete de Python y ejecutar código.

1.3. El primer programa

Tradicionalmente el primer programa en un lenguaje nuevo se llama "Hola, mundo" (Hello world!) porque sólo muestra las palabras "Hola, mundo". En Python es así:

```
>>> print('Hola, mundo')
```

Este es un ejemplo de una sentencia print, la cual no imprime nada en papel, más bien muestra un valor. En este caso, el resultado es las palabras:

```
Hola, mundo
```

Las comillas en el programa indican el inicio y el final del texto que se va a mostrar; no aparecen en el resultado. Los paréntesis indican que print es una función. Llegaremos a las funciones en el capítulo 3. En Python 2, las declaracions print es ligeremente diferente; no es una función, por lo que no usa paréntesis.

```
>>> print 'Hola, mundo'
```

Esta distición tendrá más sentido pronto, pero eso es suficiente para comenzar.

1.4. Operaciones Aritméticas

Después de "Hola, mundo", el siguiente paso es la aritmética. Python proporciona operadores, que son simbolos especiales que representan cálculos como sumas y multiplicación. Los operadores +, -, y * realizan suma, resta y multiplicación, como en los siguientes ejemplos:

```
>>> 40 + 2
42
>>> 43 - 1
```

```
42
>>> 6 * 7
42
```

El operador / realiza división:

```
>>> 84 / 2
42.0
```

Te podrías preguntar por qué el resultado es 42.0 en lugar de 42. Lo explicare en la sigueinte sección. Finalmente, el operador ** realiza exponenciación:

```
>>> 6 ** 2 + 6
42
```

En algunos lenguajes. ŝe usa para exponenciación, pero en Python es un operador a nivel de bits llamado XOR. Si no estás familiarizado con los operadores a nivel de bits, el resultado te sorprenderá:

```
>>> 6 ^ 2
4
```

No cubriré los operadores a nivel de bits en este libro, pero puedes leer sobre ellos en http://wiki.python.org/moin/BitwiseOperators

1.5. Valores y tipos

Un valor es una de las cosas básicas con las que trabaja un programa, como una letra o un número. Algunos valores que hemos visto hasta ahora son 2, 42.0, y 'Hola, Mundo!'.

Estos valores pertenecen a diferentes tipos: 2 es un entero, 42.0 es un número de punto flotante, y 'Hola, Mundo!' es una cadena, llamada así porque las letras que contiene están encadenadas juntas.

Si no estás seguro de qué tipo tiene un valor, el intérprete puede decírtelo:

```
>>> type(2)
<class 'int'>
>>> type(42.0)
<class 'float'>
>>> type('Hola, Mundo!')
<class 'str'>
```

En estos resultados, la palabra "class" se usa en el sentido de una categoría; un tipo es una categoría de valores.

No es sorprendente que los enteros pertenezcan al tipo `int`, las cadenas pertenezcan a `str` y los números de punto flotante pertenezcan a `float`.

¿Qué pasa con valores como '2' y '42.0'? Parecen números, pero están entre comillas como las cadenas.

```
>>> type('2')
<class 'str'>
>>> type('42.0')
<class 'str'>
```

Son cadenas.

Cuando escribes un entero grande, podrías sentirte tentado a usar comas entre grupos de dígitos, como en 1,000,000. Esto no es un entero legal en Python, pero es legal:

```
>>> 1,000,000
(1, 0, 0)
```

¡Eso no es para nada lo que esperábamos! Python interpreta 1,000,000 como una secuencia de enteros separados por comas. Aprenderemos más sobre este tipo de secuencia más adelante.

1.6. Lenguajes formales y lenguajes naturales

Los **lenguajes naturales** son los lenguajes hablados por seres humanos, como el español, el inglés y el francés. No los han diseñado personas (aunque se intente poner cierto orden en ellos), sino que se han desarrollado naturalmente.

Los **lenguajes formales** son lenguajes diseñados por humanos y que tienen aplicaciones específicas. La notación matemática, por ejemplo, es un lenguaje formal ya que se presta a la representación de las relaciones entre números y símbolos. Los químicos utilizan un lenguaje formal para representar la estructura química de las moléculas. Y lo más importante:

> **Los lenguajes de programación son lenguajes formales desarrollados para expresar computaciones.**

Los lenguajes formales tienden a tener reglas de sintaxis estrictas que gobiernan la estructura de las declaraciones. Por ejemplo, en matemáticas la declaración $3 + 3 = 6$ tiene sintaxis correcta, pero $3+ = 3\$6$ no la tiene. En química H_2O es una fórmula sintácticamente correcta, pero ${}_2Zz$ no lo es.

Las reglas de sintaxis vienen en dos variedades, relacionadas con *tokens* y estructura. Los tokens son los elementos básicos del lenguaje, como palabras, números y elementos químicos. Uno de los problemas con $3+ = 3\$6$ es que \$ no es un token legal en matemáticas

(al menos que yo sepa). De manera similar, 2Zz no es legal porque no hay elemento con la abreviación Zz.

El segundo tipo de regla de sintaxis se refiere a la manera en que se combinan los tokens. La ecuación 3 + /3 es ilegal porque aunque + y / son tokens legales, no puedes tener uno justo después del otro. De manera similar, en una fórmula química el subíndice viene después del nombre del elemento, no antes.

Esta es @ una oración bien estructurada en espan̈*l con t*kens inválidos en ella. Esta oración todos tokens válidos tiene, pero estructura inválida con.

Cuando lees una oración en español o una declaración en un lenguaje formal, tienes que descifrar la estructura (aunque en un lenguaje natural haces esto subconscientemente). Este proceso se llama *análisis sintáctico*.

Aunque los lenguajes formales y naturales tienen muchas características en común —tokens, estructura y sintaxis— son algunas diferencias:

ambigüedad: Los lenguajes naturales están llenos de ambigüedad, con la cual las personas lidian usando pistas contextuales y otra información. Los lenguajes formales están diseñados para ser casi o completamente no ambiguos, lo cual significa que cualquier declaración tiene exactamente un significado, independientemente del contexto.

redundancia: Para compensar la ambigüedad y reducir malentendidos, los lenguajes naturales emplean mucha redundancia. Como resultado, a menudo son verbosos. Los lenguajes formales son menos redundantes y más concisos.

literalidad: Los lenguajes naturales están llenos de modismos y metáforas. Si digo "Se me cayó la ficha", probablemente no hay ficha y nada se está cayendo (este modismo significa que alguien entendió algo después de un período de confusión). Los lenguajes formales significan exactamente lo que dicen.

Porque todos crecemos hablando lenguajes naturales, a veces es difícil ajustarse a los lenguajes formales. La diferencia entre lenguaje formal y natural es como la diferencia entre poesía y prosa, pero más acentuada:

Poesía: Las palabras se usan tanto por sus sonidos como por su significado, y todo el poema junto crea un efecto o respuesta emocional. La ambigüedad no solo es común sino a menudo deliberada.

Prosa: El significado literal de las palabras es más importante, y la estructura contribuye más significado. La prosa es más susceptible al análisis que la poesía pero aún a menudo ambigua.

Programas: El significado de un programa de computadora es no ambiguo y literal, y puede ser entendido enteramente por análisis de los tokens y estructura.

Los lenguajes formales son más densos que los lenguajes naturales, así que toma más tiempo leerlos. Además, la estructura es importante, así que no siempre es mejor leer de arriba hacia abajo, de izquierda a derecha. En su lugar, aprende a analizar el programa en tu cabeza, identificando los tokens e interpretando la estructura. Finalmente, los detalles importan. Pequeños errores en ortografía y puntuación, con los cuales te puedes salir con la tuya en lenguajes naturales, pueden hacer una gran diferencia en un lenguaje formal.

1.7. Depuración

Los programadores cometen errores. Por razones caprichosas, los errores de programación se llaman *bugs* (errores) y el proceso de rastrearlos se llama *depuración* (debugging).

La programación, y especialmente la depuración, a veces saca emociones fuertes. Si estás luchando con un error difícil, podrías sentirte enojado, desalentado o avergonzado.

Hay evidencia de que las personas naturalmente responden a las computadoras como si fueran personas. Cuando funcionan bien, pensamos en ellas como compañeros de equipo, y cuando son obstinadas o groseras, respondemos a ellas de la misma manera que respondemos a personas groseras y obstinadas (Reeves y Nass, *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*).

Prepararse para estas reacciones podría ayudarte a lidiar con ellas. Un enfoque es pensar en la computadora como un empleado con ciertas fortalezas, como velocidad y precisión, y debilidades particulares, como falta de empatía e incapacidad para captar el panorama general.

Tu trabajo es ser un buen gerente: encontrar maneras de aprovechar las fortalezas y mitigar las debilidades. Y encontrar maneras de usar tus emociones para involucrarte con el problema, sin dejar que tus reacciones interfieran con tu capacidad de trabajar efectivamente.

Aprender a depurar puede ser frustrante, pero es una habilidad valiosa que es útil para muchas actividades más allá de la programación. Al final de cada capítulo hay una sección, como esta, con mis sugerencias para depuración. ¡Espero que ayuden!

1.8. Glosario

resolución de problemas: El proceso de formular un problema, encontrar una solución y expresarla.

lenguaje de alto nivel: Un lenguaje de programación como Python que está diseñado para ser fácil de leer y escribir para los humanos.

lenguaje de bajo nivel: Un lenguaje de programación que está diseñado para ser fácil de ejecutar para una computadora; también llamado “lenguaje de máquina” o

"lenguaje ensamblador".

portabilidad: Una propiedad de un programa que puede ejecutarse en más de un tipo de computadora.

intérprete: Un programa que lee otro programa y lo ejecuta.

prompt: Caracteres mostrados por el intérprete para indicar que está listo para recibir entrada del usuario.

programa: Un conjunto de instrucciones que especifica una computación.

declaración print: Una instrucción que hace que el intérprete de Python muestre un valor en la pantalla.

operador: Un símbolo especial que representa una computación simple como suma, multiplicación o concatenación de cadenas.

valor: Una de las unidades básicas de datos, como un número o cadena, que un programa manipula.

tipo: Una categoría de valores. Los tipos que hemos visto hasta ahora son enteros (tipo `int`), números de punto flotante (tipo `float`) y cadenas (tipo `str`).

entero: Un tipo que representa números enteros.

punto flotante: Un tipo que representa números con partes fraccionarias.

cadena: Un tipo que representa secuencias de caracteres.

lenguaje natural: Cualquiera de los lenguajes que hablan las personas que evolucionaron naturalmente.

lenguaje formal: Cualquiera de los lenguajes que las personas han diseñado para propósitos específicos, como representar ideas matemáticas o programas de computadora; todos los lenguajes de programación son lenguajes formales.

token: Uno de los elementos básicos de la estructura sintáctica de un programa, análogo a una palabra en un lenguaje natural.

sintaxis: Las reglas que gobiernan la estructura de un programa.

analizar: Examinar un programa y analizar la estructura sintáctica.

bug: Un error en un programa.

depuración: El proceso de encontrar y corregir bugs.

1.9. Ejercicios

Ejercicio 1.1. Es una buena idea leer este libro frente a una computadora para que puedas probar los ejemplos mientras avanzas. Siempre que estés experimentando con una nueva característica, deberías tratar de cometer errores. Por ejemplo, en el programa "¡Hola, mundo!", ¿qué pasa si omites una de las comillas? ¿Qué pasa si omites ambas? ¿Qué pasa si escribes `print` mal? Este tipo de experimento te ayuda a recordar lo que lees; también te ayuda cuando estás programando, porque llegas a saber qué significan los mensajes de error. Es mejor cometer errores ahora y a propósito que después y accidentalmente.

1. En una declaración `print`, ¿qué pasa si omites uno de los paréntesis, o ambos?
2. Si estás tratando de imprimir una cadena, ¿qué pasa si omites una de las comillas, o ambas?
3. Puedes usar un signo menos para hacer un número negativo como `-2`. ¿Qué pasa si pones un signo más antes de un número? ¿Qué tal `2++2`?
4. En notación matemática, los ceros iniciales están bien, como en `09`. ¿Qué pasa si intentas esto en Python? ¿Qué tal `011`?
5. ¿Qué pasa si tienes dos valores sin operador entre ellos?

Ejercicio 1.2. Inicia el intérprete de Python y úsalo como calculadora.

1. ¿Cuántos segundos hay en 42 minutos 42 segundos?
2. ¿Cuántas millas hay en 10 kilómetros? Pista: hay 1.61 kilómetros en una milla.
3. Si corres una carrera de 10 kilómetros en 42 minutos 42 segundos, ¿cuál es tu ritmo promedio (tiempo por milla en minutos y segundos)? ¿Cuál es tu velocidad promedio en millas por hora?