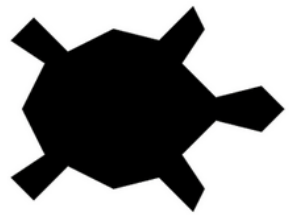
Introducción a la Programación

Prof. Agustín Gravano

Primer semestre de 2022

Clase teórica 24: Recursión algorítmica

Biblioteca `turtle` de Python



```
import turtle as t
```

Instrucciones principales:

- `t.forward(d)`: avanzar la tortuga la distancia `d`.
- `t.backward(d)`: retroceder la tortuga la distancia `d`.
- `t.left(a)`: rotar la tortuga `a` grados hacia la izquierda.
- `t.right(a)`: rotar la tortuga `a` grados hacia la derecha.
- `t.mainloop()` y `t.bye()`: siempre incluir al final del programa.

Otras instrucciones:

- `t.home()`: llevar la tortuga al origen (0,0).
- `t.goto(x,y)`: llevar la tortuga a la posición `(x,y)`.
- `t.reset()`: limpiar la pantalla y volver al origen.
- `t.penup()`: levantar el lápiz, para moverse sin escribir.
- `t.pendown()`: bajar el lápiz, para escribir al moverse (por defecto, el lápiz está bajo: la tortuga va escribiendo).

Documentación: `http://docs.python.org/library/turtle.html`

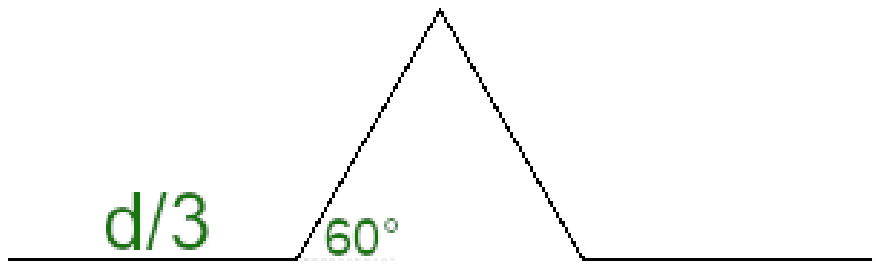
Ejemplos

```
1  import turtle as t
2
3  def triangulo(d:float):
4      ''' Dibuja un triángulo equilátero de lado d. '''
5      t.forward(d)
6      t.right(120)
7      t.forward(d)
8      t.right(120)
9      t.forward(d)
10
11 def lineas_punteadas(n:int, d:float):
12     ''' Dibuja n líneas punteadas de longitud d. '''
13     for i in range(n):
14         t.pendown()
15         t.forward(d) # dibujo
16         t.penup()
17         t.forward(d) # no dibujo
18
19 triangulo(100)
20 lineas_punteadas(20, 5)
21
22 t.mainloop()          # Siempre incluir estas dos operaciones
23 t.bye()               # al final del programa.
```

Ejercicios (por ahora sin recursión)

Definir las siguientes funciones:

1. `cuadrado(d:float)`, que dibuja un cuadrado de lado `d`.
2. `montaña(d:float)`, que dibuja una montaña como la siguiente:



Instrucciones principales:

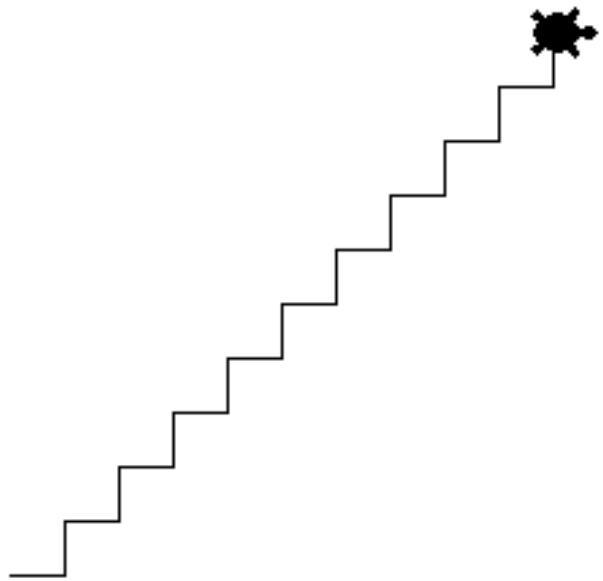
- `t.forward(d)`: avanzar la tortuga la distancia `d`.
- `t.left(a)`: rotar la tortuga `a` grados hacia la izquierda.
- `t.right(a)`: rotar la tortuga `a` grados hacia la derecha.

Con `t.speed(0)` podemos pedirle a la tortuga que vaya a la máxima velocidad posible. (Pero no le pidamos mucho; sigue siendo una tortuga.)

Ejercicio (ahora con recursión)

Definir una función `escalera(d:float, n:int)`, que dibuje una escalera con `n` escalones de largo y alto `d`.

Ejemplo: `escalera(20, 10)`



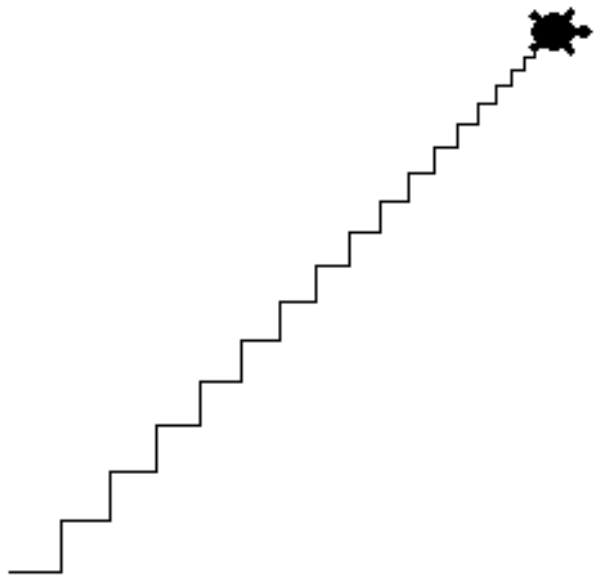
Hacer la recursión sobre la cantidad de escalones (`n`).

¿Cómo serían ser el caso base (cuando ya terminamos de dibujar) y el caso recursivo (cuando todavía nos faltan algunos escalones)?

Ejercicio (con recursión)

Definir una función `escalera2(d:float, k:float)`, que dibuje una escalera que empieza con escalones de largo y alto `d`, restando `k` a cada nuevo escalón hasta llegar a 0.

Ejemplo: `escalera2(20, 1)`

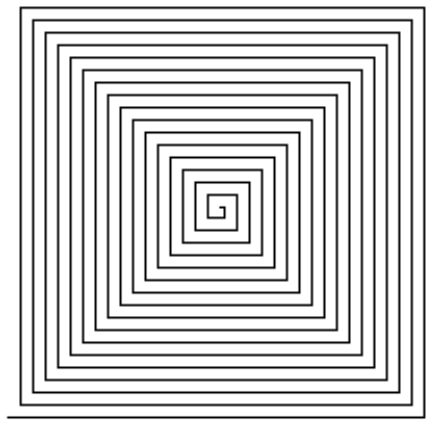


Hacer la recursión sobre el tamaño del escalón (`d`).

¿Cómo serían ser el caso base (cuando ya terminamos de dibujar) y el caso recursivo (cuando todavía nos faltan algunos escalones)?

Ejercicio

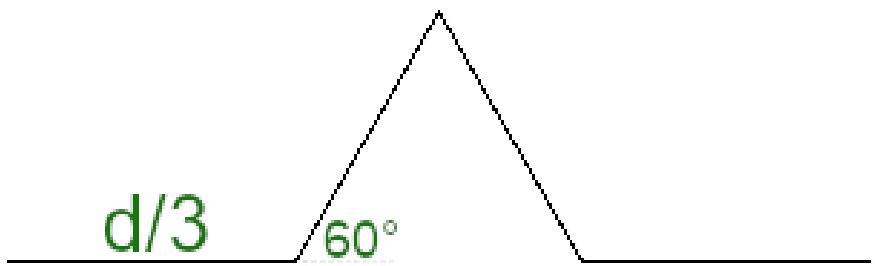
Definir una función recursiva `espiral_cuadrada(d:float, k:float)`, que dibuje una espiral cuadrada como esta:



El parámetro `d` es el largo del lado más grande de la espiral, y `k` es cuánto se reduce cada lado respecto del anterior.

Montaña simple

```
1  import turtle as t
2  def montaña(d:float):
3      ''' Dibuja una montaña simple de lado d. Requiere: d>0 '''
4      t.forward(d/3)
5      t.left(60)
6      t.forward(d/3)
7      t.right(120)
8      t.forward(d/3)
9      t.left(60)
10     t.forward(d/3)
```



En la función `montaña`, vamos a reemplazar cada trazo de longitud `d/3` por un llamado recursivo a `montaña`...

Montaña recursiva (versión 1)

```
1  import turtle as t
2  def montaña_recursiva(d:float):
3      if d<5:        # caso base
4          t.forward(d)
5      else:
6          montaña_recursiva(d/3)
7          t.left(60)
8          montaña_recursiva(d/3)
9          t.right(120)
10         montaña_recursiva(d/3)
11         t.left(60)
12         montaña_recursiva(d/3)
```

El dibujo resultante se llama curva de Koch. Si reemplazamos los lados de un triángulo por curvas de Koch, obtenemos un copo de nieve de Koch:

```
1  montaña_recursiva(d)
2  t.right(120)
3  montaña_recursiva(d)
4  t.right(120)
5  montaña_recursiva(d)
```

Montaña recursiva (versión 2)

```
1  import turtle as t
2  def montaña_recursiva(d:float, nivel:int):
3      if nivel==0:        # caso base
4          t.forward(d)
5      else:
6          montaña_recursiva(d/3, nivel-1)
7          t.left(60)
8          montaña_recursiva(d/3, nivel-1)
9          t.right(120)
10         montaña_recursiva(d/3, nivel-1)
11         t.left(60)
12         montaña_recursiva(d/3, nivel-1)
```

El dibujo resultante se llama curva de Koch. Si reemplazamos los lados de un triángulo por curvas de Koch, obtenemos un copo de nieve de Koch:

```
1  montaña_recursiva(d, 4)
2  t.right(120)
3  montaña_recursiva(d, 4)
4  t.right(120)
5  montaña_recursiva(d, 4)
```

Repaso de la clase de hoy

- Recursión algorítmica, ahora en forma gráfica con `turtle`
- Caso base y caso recursivo de las funciones recursivas

Bibliografía complementaria:

- APPP2, secciones 4.9 a 4.11, 5.5 a 5.8.
- HTCSP3, capítulo 18.
- Documentación y tutoriales de `turtle`:
  - `https://docs.python.org/library/turtle.html`
  - `https://opentechschool.github.io/python-beginners/es_CL/simple_drawing.html`
  - `https://www.programoergosum.es/tutoriales/introduccion-a-turtle-python-en-raspberry-pi/`

Con lo visto, ya pueden resolver la Guía de Ejercicios 10 completa.

TD1: Introducción a la Programación · Epílogo

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.

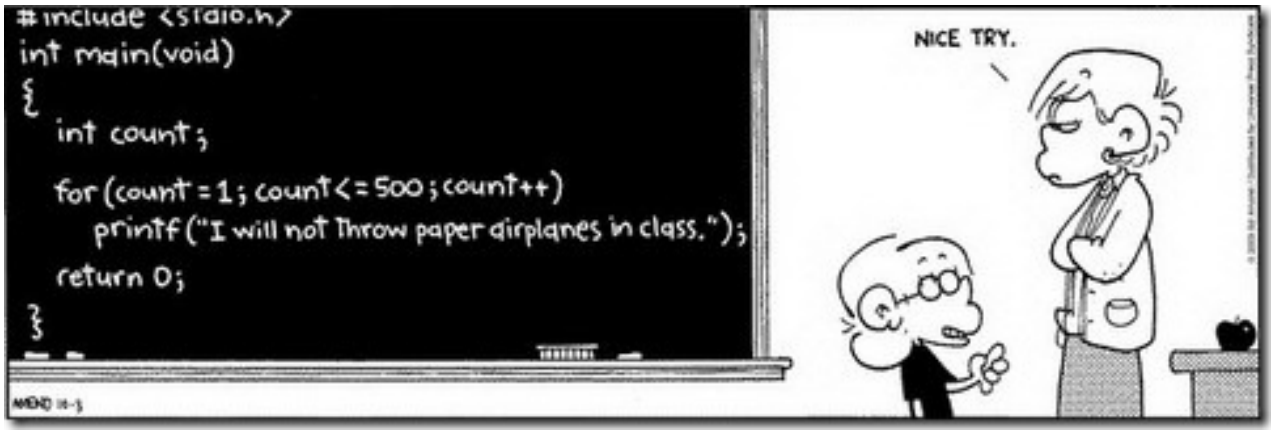
Conceptos independientes del lenguaje de programación.

Punto de partida para que explorar y seguir aprendiendo.

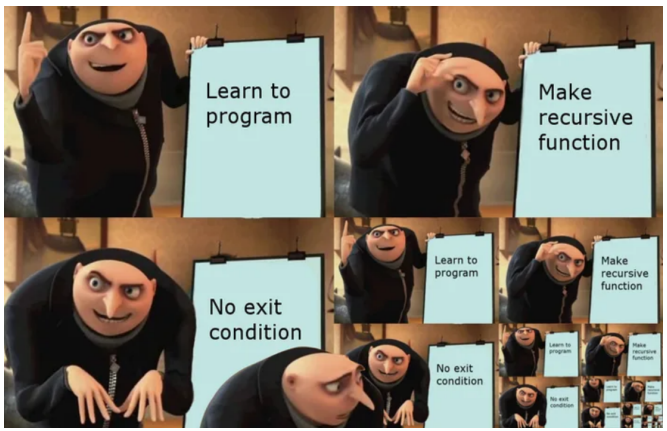
Hagan un esfuerzo por mantener las buenas prácticas de programación.

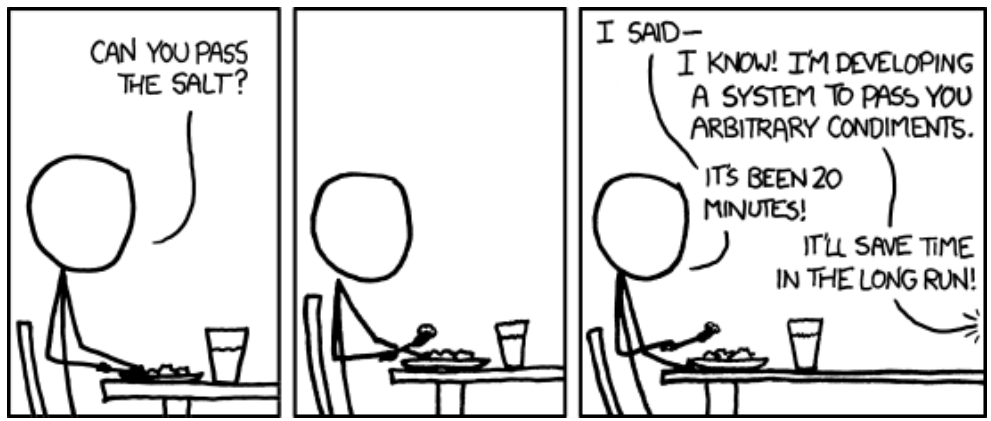
TD1: Introducción a la Programación · Epílogo

¿Para qué sirvió esta materia? ¡Para entender chistes nerds! :-)



¿Por qué murió el programador en la ducha? Porque siguió las instrucciones de la botella de shampoo: *Aplicar. Enjuagar. Repetir.*





https://xkcd.com/974/