Programação II

Iteração, condicionais e funções

Hugo Pacheco

DCC/FCUP
20/21

Fluxo de programas

- Python é uma linguagem imperativa
- Programa definido como sequência de instruções
- Tipicamente 1 linha = 1 instrução, mas nem sempre

```
print(3)
print(4)

print(3); print(4)
```

- Fluxo do programa = instrução a instrução, de cima para baixo

Um programa Turtle

- módulo turtle
- configuração da janela
- configuração da tartaruga
- movimento da tartaruga
- fazer janela esperar

```
import turtle

window = turtle.Screen()
window.bgcolor("lightgreen")
window.title("Hello, Alex!")

alex = turtle.Turtle()
alex.color("blue")
alex.pensize(3)

alex.forward(50)
alex.left(120)
alex.forward(50)

#window.mainloop()
```

Um programa Turtle

- módulo *turtle*

- objectos (têm estado interno “escondido”)

  window : Screen

  alex : Turtle

- Métodos (alteram o estado interno)

  objeto.método(args)

```
import turtle

window = turtle.Screen()
window.bgcolor("lightgreen")
window.title("Hello, Alex!")

alex = turtle.Turtle()
alex.color("blue")
alex.pensize(3)

alex.forward(50)
alex.left(120)
alex.forward(50)

#window.mainloop()
```

Iteração (ciclo for)

- Um dos elementos principais em programação é a repetição de instruções = iteração

- Uma das formas de iteração mais simples é o ciclo **for** (indentação importante)

```
for friend in ["Joe", "Zoe", "Zuki", "Thandi", "Paris"]:
    invite = "Hi " + friend + ". Please come to my party!"
    print(invite)

for x in "banana":
  print(x)

for x in range(6):
  print(x)

for x in ["red", "big", "tasty"]:
  for y in ["apple", "banana", "cherry"]:
    print(x, y)
```

Iteração (ciclo for + turtle)

- Desenhar um quadrado

```
alex.forward(100)
alex.left(90)
alex.forward(100)
alex.left(90)
alex.forward(100)
alex.left(90)
alex.forward(100)
alex.left(90)
```

- Repetição de padrões

```
for _ in range(4):
    alex.forward(100)
    alex.left(90)
```

Booleanos

- Um **bool** é **True** ou **False**
- casts
- comparação
- lógica booleana

```
print(bool("Hello"))
print(bool(15))
print(bool(""))
print(bool(0))

print(10 == 9)
print (10 != 9)
print(10 > 9)
print(9 >= 9)
print(10 < 9)
print(10 <= 9)

print(x < 5 and x < 10)
print(x < 5 or x < 4)
print(not(x < 5 and x < 10))
```

Condicionais

- Um dos elementos principais em programação é o comportamento por casos = condicionais

```
if x % 2 == 0: print(x,"is even")
else: print(x,"is odd")
```

```
print(x,"is even") if x % 2 else print(x,"is odd")

if x > y: print(“x is greater than y")
elif x == y: print("x and y are equal")
else: print("y is greater than x”)

if 0 < x:
    if x < 10:
        print("x is a positive single digit.")

if 0 < x and x < 10:
    print("x is a positive single digit.")
```

Ciclo for

- Permite repetir instruções um <u>número fixo</u> de vezes
- Percorrer um **iterador**, i.e., uma sequência de elementos
- Pode ser uma string, um range numérico, uma lista, etc

- E.g., somar uma lista de números inteiros

```
numbers = [5, 6, 32, 21, 9]
running_total = 0
for number in numbers:
    running_total += number
print(running_total)
```

Ciclo while

- Permite repetir instruções um <u>número indefinido</u> de vezes, controlado dinamicamente pelo próprio ciclo

- E.g., somar uma lista de números inteiros (inicialização, condição do ciclo, atualização)

```
numbers = [5, 6, 32, 21, 9]
running_total = 0
i = 0;
while (i < len(numbers)):
    running_total += numbers[i]
    i+=1
print(running_total)
```

Ciclo + condicional

- Um ciclo pode ter um corpo condicional

```
xs = [12, 16, 17, 24, 29]

for x in xs:
    if x % 2 == 0: print(x)

for x in xs:
    if x % 2 == 0: print(x)
    else: break;

i=0
while (i < len(xs) and xs[i] % 2 == 0):
    print(xs[i]); i+=1
```

Ciclo while (Collatz)

- Função de Collatz

$$f(n) = \begin{cases} n/2, & n \textbf{ par} \\ 3n+1, & n \textbf{ ímpar} \end{cases}$$

- Calcular sequência de Collatz para n > 0 enquanto n != 1

```
while n != 1:
    if n%2 == 0: n = n//2
    else: n = 3*n+1
    print(n)
```

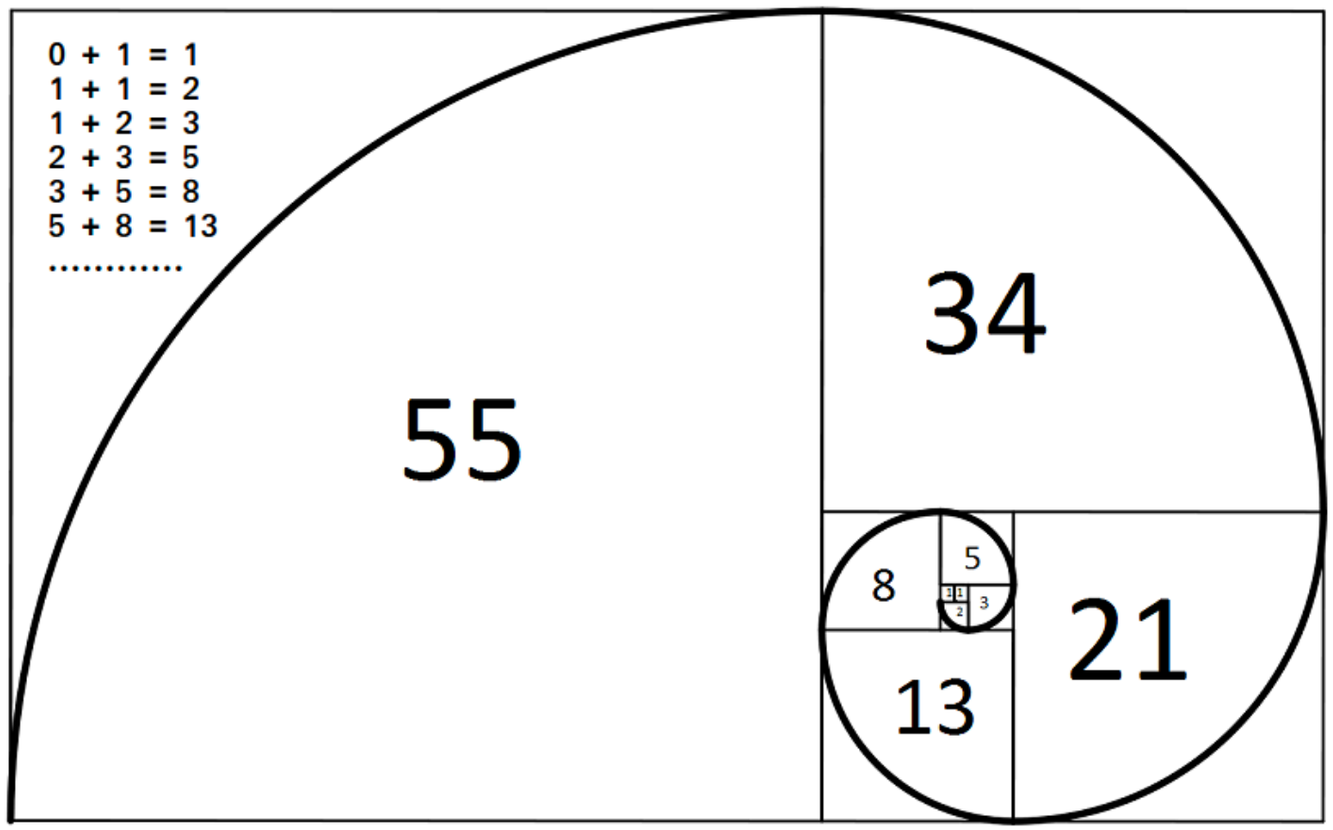
- Nota: não sabemos se esta função termina para todo o n > 0!

Ciclo while (Fibonnaci)

- Função de Fibonnaci

$$f(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ f(n-1) + f(n-2), & n > 1 \end{cases}$$

- Desenhar sequência de Fibonnaci



Ciclo while (fib + turtle)

```
import turtle
import math

window = turtle.Screen(); alex = turtle.Turtle()
prev = 0; start = 1; fib = 1

alex.right(90)
while True:
    arc = math.pi * start / 10
    for j in range(90):
        alex.forward(arc)
        alex.left(1)
    fib = start + prev; prev = start; start = fib
```

Funções

- Um dos elementos principais em programação é decomposição de problemas = funções

- Analogia com funções matemáticas, recebem inputs e retornam outputs

- Suportam documentação especial *docstring*

- E.g., segunda lei de Newton para queda de corpos em função da velocidade inicial *v0 e* tempo *t, com* constante gravítica *g*

```
def y(v0,t):
    """calcula posição vertical de um corpo"""
    g = 9.8
    return v0 * t - 1 / 2 * g * (t**2)
help(y)
y(1,2)
```

Funções (fluxo)

- Definição de funções não altera fluxo do programa
- Pode definir-se funções no meio do código, mas não é recomendado
- Função apenas é executada quando chamada, fluxo salta para a 1ª linha da função, e retorna ao ponto onde estava pós saída da função
- Função pode utilizar parâmetros globais
- Função tem parâmetros e variáveis locais, destruídos à saída

```
g = 9.8
def y(v0, t):
    return v0 * t - 1 / 2 * g * t * t
v0=1
print("t=0:",y(v0,0))
print("t=2:",y(v0,2))
```

Funções ou Procedimentos

- Funções matemáticas são puras, i.e., só dependem dos seus argumentos e não têm efeitos laterais (à la programação funcional, **preferível**)

- Funções Python podem não ser puras (chamadas de procedimentos), por exemplo, quando nem têm valor de retorno (só justificável por questões de eficiência)

```
def f(x):
    y = x**2
    print(y)
print(f(2))
```

```
y = 0
def f(x):
    y = x**2
print(y)
```

Funções (composição)

- Funções podem ser compostas, i.e., chamar outras funções
- Funções podem definir sub-funções locais (não recomendado)
- E.g., calcular a área de um círculo com raio igual à distância entre dois pontos

```
import math
def distance(x1, y1, x2, y2):
    return math.sqrt( (x2-x1)**2 + (y2-y1)**2 )
def area(radius):
    return math.pi * radius**2
def area_of_circle(xc, yc, xp, yp):
    return area(distance(xc, yc, xp, yp))
print(area_of_circle(0, 0, 0, 1))
```

Funções (turtle)

- Desenhar uma espiral de quadrados multicolores

```
import turtle

def draw_multicolor_square(animal,size):
    for color in ["red", "purple", "hotpink", "blue"]:
        animal.color(color)
        animal.forward(size)
        animal.left(90)

window = turtle.Screen(); alex = turtle.Turtle()

size = 20
while True:
    draw_multicolor_square(alex,size)
    size += 10
    alex.forward(10)
    alex.right(18)

window.mainloop()
```