Clase 08 - Control de flujo cíclico (II)

IIC1103-07 - Introducción a la Programación

Cristian Ruz – cruz@ing.puc.cl

Martes 3-Septiembre-2019

Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile

Contenidos

Contenidos

Laboratorios: Lab 1 y Lab 2 disponibles

- Laboratorios evaluados automáticamente (*hackerrank*)
- Pueden ir a cualquier laboratorio (Lunes a Jueves)
- Pueden hacerlos **fuera de la sala** (Lunes a Domingo)
- No doy puntos por asistencia, pero les recomiendo ir
  - SIN LAPTOP
    - Lunes a Jueves, mód 5 y 6: Lab San Agustín (piso 2)
  - CON LAPTOP
    - Lunes, mód 5 y 6: CS204
    - Martes, mód 5 y 6: K200, A5, B12
    - Miércoles, mód 5 y 6: **B23**, CS203, C203
    - Jueves, mód 5 y 6: B13, K204, CS101
- **¡Este es un curso práctico! Aprender haciendo**
- **¡¡LABS 1 y 2 ya están disponibles!!**
- **Lunes 23-Septiembre, 18:00**, se recolecta puntaje de Labs 1 al 4.

Contenidos

Hackerrank de Contol de Flujo Condicional

¿Cómo les fue con el *Hackerrank* del viernes 30?

Puntajes en la planilla de la sección.

Contenidos

¿En qué vamos?

1. **Variables y expresiones**
2. **Control de flujo** $\Leftarrow$ *Aquí vamos*
3. Funciones y recursión $\Leftarrow$ *I1: 24-Sept, 18:30*
4. Strings
5. Listas
6. Tipos de datos personalizados (objetos)
7. Ordenación y búsqueda
8. Archivos

Ciclos (loops): `while`

Instrucción `while`

Permite ejecutar varias veces la misma sección de código

```
while condicion:              # Siempre debe estar
  bloque_de_codigo_while      # Este bloque se repite
  ...                         # MIENTRAS se cumpla la
  ...                         # condicion
  ...
  ...                         # Despues de esto se vuelve
  bloque_de_codigo_while      # a comprobar la condicion

codigo_fuera_de_while         # Esto se ejecuta cuando
codigo_fuera_de_while         # la condicion
                              # deja de cumplirse
```

Ejercicio 1

Código que pide un número n al usuario, e imprime todos los números entre 1 y n

```
1 n = int(input("Ingrese un numero entero: "))
2 i = 1
3 while i<=n:
4   print(i)
5   i = i + 1
```

Ejercicio 2

Código que pide un número *n* al usuario, e imprima todos los números *pares* entre 1 y *n* (1 línea más)

```
1 n = int(input("Ingrese un numero entero: "))
2 i = 1
3 while i<=n:
4   if (i%2)==0:    #Unica linea extra
5     print(i)      #Recuerden indentar esta linea
6   i = i + 1
```

Ejercicio 2

Código que pide un número *n* al usuario, e imprima todos los números *pares* entre 1 y *n* (alternativa)

```
1 n = int(input("Ingrese un numero entero: "))
2 i = 2              #Empieza con un numero par
3 while i<=n:
4   print(i)         #Imprimo todos
5   i = i + 2        #Pero ahora sumo 2
```

Ejercicio 3

Código que pide un número n al usuario, e indique si el número es primo. El programa debe terminar si el usuario ingresa 1

Ejercicio 3

Código que pide un número n al usuario, e indique si el número es primo. Debe terminar si $n \leq 1$

```
#Primera idea
n = int(input("Ingrese un numero entero:"))
if n<=1:
  print(n,"no es primo ni compuesto")
else:
  esPrimo = True  #Supongamos que n es primo
  if esPrimo:
    print(n, "es primo")
  else:
    print(n, "es compuesto")
```

¿Cuándo n es primo? Cuando n es divisible **SOLAMENTE** por 1 y por n.

Ejercicio 3: ¿Cuándo un número es primo?

Cuando *n* es divisible **SOLAMENTE** por 1 y por *n*.

```python
#Segunda idea
n = int(input("Ingrese un numero entero:"))
if n<=1:
  print(n,"no es primo ni compuesto")
else:
  esPrimo = False  #Supongamos que n es primo
  if (n%1)==0 and (n%n)==0:
    esPrimo = True
  if esPrimo:
    print(n, "es primo")
  else:
    print(n, "es compuesto")
```

¿Qué problema tiene esta solución?

Ejercicio 3: ¿Cuándo un número es primo?

Necesitamos que **NO SEA DIVISIBLE** por algún i desde 2 a n[1]

```python
#Tercera idea (la vencida)
n = int(input("Ingrese un numero entero: "))
if n<=1:
  print(n,"no es primo ni compuesto")
else:
  esPrimo = True  #Supongamos que n es primo
  i = 2
  while i<n:
    if (n%i)==0:  #Si es divisible por i
      esPrimo = False  #ya no es primo
    i = i+1
  if esPrimo:
    print(n, "es primo")
  else:
    print(n, "es compuesto")
```

[1] Puede ser solo hasta $\sqrt{n}$

Ejercicio 3: Para pensar ...

¿Puede ser más rápido?

```python
#Version reducida
n = int(input("Ingrese un numero entero: "))
if n<=1: print(n,"no es primo ni compuesto")
else:
  esPrimo = True  #Supongamos que n es primo
  i = 2
  while i<n:
    if (n%i)==0:  #Si es divisible por i
      esPrimo = False  #ya no es primo
    i += 1
  if esPrimo: print(n, "es primo")
  else: print(n, "es compuesto")
```

¿Necesito probar todos los números si ya no es primo?

¿Cuánto demora en verificar si 179426549 es primo? (lo es)

Código de marcador de fútbol

```
local = 0          # goles del local
visita = 0         # goles de la visita
# [1: local, 2: visita, otro: fin partido]
gol = int(input("¿Qué equipo anotó el gol?"))
print("Gol del equipo: ", gol)

local = local + 1      # gol local
visita = visita + 1    # gol visita

print("Local", local, "-", visita, "Visita")
# Ahora veo quien gana
if local > visita:
  print("¡Ganó el local!")
else if local < visita:
  print("¡Ganó la visita!")
else:
  print("¡Empate!")
```


¿Cómo aprovechamos los ciclos para mejorar este simulador?

Contenidos

Cuando queremos contar una cantidad fija de números

```
for variable in range(inicial,final):
  bloque_de_código_for
  ...
  bloque_de_código_for
bloque_de_código_fuera_del_for
```

Permite repetir una cantidad: `final-inicial` veces.

Equivalente

Contando con `while`

```
1 i = 5
2 while i < 15:
3   print(i)
4   i = i + 1
```

Contando con `for`

```
1 for i in range(5,15):
2   print(i)
```

`range` recorrer desde `inicial` hasta `final-1`

Contenidos

Rompiendo ciclos

Es posible terminar un ciclo sin verificar la condición del `while` (¡trampa!)

`break`
Permite salir del ciclo en cualquier momento

```
1 i = 0
2 while True:    #ciclo infinito
3   i = i + 1
4   print("Ciclo",i)
```

Este es un ciclo infinito.

Rompiendo ciclos

break
Permite salir del ciclo en cualquier momento

Puedo querer salir cuando *i* sea 42

```
1 i = 0
2 while True :  #ciclo infinito
3   i = i + 1
4   if i==42:
5     break     #salimos del ciclo
6   print("Ciclo",i)
7 print("Fuera del ciclo",i)      #¿Qué valor imprime?
```

Rompiendo ciclos

Sintaxis `continue`

Regresa a la verificación de la condición, ignorando el resto del ciclo actual, pero **sin salir del `while`**.

```
1 i = 0
2 while i<42 :
3   i = i + 1
4   if i%7 == 0:
5     continue          #Volvemos a la linea 2
6   print("Ciclo",i)
```

¿Qué números no imprime?

Rompiendo ciclos

Podemos mezclar `break` y `continue`

```
1 i = 0
2 while True:           #ciclo infinito
3   i = i + 1
4   if i%5==0:
5     continue          #salto los divisibles por 5
6   if i==42:
7     break             #salgo del ciclo
8   print("Ciclo",i)
9 print("Fuera del ciclo", i)
```

Podemos usar break

Podemos usar **break** para los primos

```
#Cuarta idea (más rápida)
n = int(input("Ingrese un numero entero: "))
if n<=1:
  print(n,"no es primo ni compuesto")
else:
  esPrimo = True  #Supongamos que n es primo
  i = 2
  while i<n:
    if (n%i)==0:  #Si es divisible por i
      esPrimo = False  #ya no es primo
      break            #no sigo revisando
    i = i+1
  if esPrimo:
    print(n, "es primo")
  else:
    print(n, "es compuesto")
```

O podemos no usar break

Pero también podríamos haberlo hecho así:

```
#Cuarta idea (sin break)
n = int(input("Ingrese un numero entero: "))
if n<=1:
  print(n,"no es primo ni compuesto")
else:
  esPrimo = True  #Supongamos que n es primo
  i = 2
  while i<n and esPrimo: #ok, sin break
    if (n%i)==0:  #Si es divisible por i
      esPrimo = False  #ya no es primo
    i = i+1
  if esPrimo:
    print(n, "es primo")
  else:
    print(n, "es compuesto")
```