# ¿Podemos programar únicamente con bucles `for`?

Antonio Casares Santos

ValllaENEM 2020

# Un primer ejemplo

*Input*: $n \in \mathbb{N}$, $n \geq 2$.

*Problema*: Encontrar el primer divisor de $n$ mayor que 1.

```
Function BuscarDivisor(n)

encontrado:=0
d:=2
While encontrado==0
  if d divide n
    encontrado==1
  else
    d:=d+1
return d
```

```
Function BuscarDivisor2(n)

For d from 2 to n
  if d divide n
    return d
```

¿Podemos eliminar siempre los bucles `While`?

# ¿Por qué queremos eliminar los bucle `While`?

➔ Los bucles `While` crean problemas desde un punto de vista teórico.

```
While 1+1==2
  a:=1
```

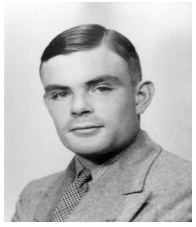
**Programa que no termina.**

```
n:=0
test:=1
While test==1
  n:=n+1
  test:=0
  For p,q primos menores que n:
    if p+q=n
      test:=1

print("La conjetura de Goldbach es falsa!")
```

**¿Termina este programa?**

## Los comienzos de la computación


Alan Turing


Alonzo Church

En la década de 1930 se proponen distintos modelos de computación.

### Función computable

Función $f : \mathbb{N}^k \to \mathbb{N}$ para la que podemos dar una lista de instrucciones precisas (un programa) tal que tras aplicar estas instrucciones a un $\vec{n} \in \mathbb{N}^k$, obtenemos $f(\vec{n})$.

➔ Suponemos que tenemos todo el tiempo y memoria que necesitemos.

### Observación

*Existen (muchas!) funciones que no son computables.*

*Hay $2^{\aleph_0}$ funciones $f : \mathbb{N} \to \mathbb{N}$, pero sólo un número numerable de programas que podemos escribir.*

# Un modelo sin bucles `While`: funciones recursivas primitivas

Thoralf Skolem

Kurt Gödel

Rózsa Péter

Funciones base:

*Porque... ¡sí, $0 \in \mathbb{N}$!*

- El 0
- Función sucesor $s(n) = n + 1$
- Igualdad: *Igual*$(n, m)$ es 1 si $n = m$, 0 en caso contrario.

Método de iteración:

- Bucles **`For`** (¡sin modificar el índice de recursión!).

➔ Copiar y pegar programas → Composición de funciones

# Funciones que obtenemos

➔Todas las constantes $c \in \mathbb{N}$ : $4 = s(s(s(s(0))))$.

```
Function Suma(n,m)

suma:=n
For i from 1 to m
  suma:= s(suma)

return suma
```

```
Function Multiplicacion(n,m)

mult:=0
For i from 1 to m
  mult:= mult+n

return mult
```

# Funciones que obtenemos

➔Todas las constantes $c \in \mathbb{N}$ : $4 = s(s(s(s(0))))$.

```
Function Suma(n,m)

suma:=n
For i from 1 to m
   suma:= s(suma)

return suma
```

```
Function Multiplicacion(n,m)

mult:=0
For i from 1 to m
   #mult:= mult+n
   suma:=mult
   For i from 1 to m
      suma:= s(suma)
   mult:=suma
return mult
```

Bucles For Imbricados

# Funciones que obtenemos

➔Todas las constantes $c \in \mathbb{N}$ : $4 = s(s(s(s(0))))$.

```
Function Suma(n,m)

suma:=n
For i from 1 to m
suma:= s(suma)

return suma
```

```
Function Multiplicacion(n,m)

mult:=0
For i from 1 to m
mult:= mult+n

return mult
```

➔Exponenciación(n,m): añadir un bucle For.

```
exp:=1
For i from 1 to m
  exp:= exp · n
```

➔Podemos programar condicionales if, comparaciones $\leq$, y definir funciones por casos.

¿Obtenemos todas las funciones que desearíamos?

# La función de Ackermann

Wilhelm Ackermann

## Definición (Función de Ackermann[1])

$Ack : \mathbb{N}^2 \to \mathbb{N}$

$$Ack(0,n) = n+2$$
$$Ack(1,0) = 0$$
$$Ack(k,0) = 1 \quad para\ k > 1$$
$$Ack(k,n) = Ack(k-1, Ack(k,n-1)) \quad si\ \ k > 0\ \ y\ \ n > 0$$

$$Ack(1,2) = Ack(0, Ack(1,1)) = Ack(0, Ack(0, Ack(1,0))) =$$
$$= Ack(0, Ack(0,0)) = Ack(0,2) = 4$$

[1] Versión de Rózsa Péter obtenida de apuntes de Paul Rozière (IRIF).

$$Ack(0,n) = n+2$$
$$Ack(1,0) = 0$$
$$Ack(k,0) = 1 \quad k > 1$$
$$Ack(k,n) = Ack(k-1, Ack(k,n-1))$$

$$Ack_k(n) = Ack(k,n)$$

0 Bucles `For`

$$Ack_0(n) = n+2$$

$$Ack_1(n) = Ack_0(Ack_1(n-1)) = Ack_1(n-1) + 2$$

1 Bucle `For`

$$Ack_1(n) = 2 \cdot n$$

$$Ack_2(n) = Ack_1(Ack_2(n-1)) = 2 \cdot Ack_2(n-1)$$

2 Bucles `For` imbricados

$$Ack_2(n) = 2^n$$

$Ack(3,5) \geq$ Número de partículas en el universo observable... ¡elevado a 100!!

$$Ack_3(n) = 2^{Ack_3(n-1)}$$

3 Bucles `For` imbricados

$$Ack_3(n) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \quad \text{(n)}$$

## Jerarquía para las funciones definidas con bucles `For`

### Proposición

*Si una función $f : \mathbb{N} \to \mathbb{N}$ se define con un programa usando menos de $k$ bucles `For` imbricados, entonces*

$$f(n) < Ack_{k+1}(n)$$

*para $n$ suficientemente grande.*

### Función diagonal

$$D(n) = Ack(n, n)$$

### Teorema

*La función $D(n)$ no puede ser computada por un programa utilizando únicamente bucles `For`.*

## Demostración

### Teorema

*La función $D(n) = Ack(n, n)$ no puede ser computada por un programa utilizando únicamente bucles* `For`.

### Proof.

- Si $D(n)$ fuese computada por un programa, éste utilizaría un máximo de $k$ bucles `For` imbricados, para un $k$ fijo.
- $D(n) < Ack_{k+1}(n)$

Imposible, pues para $n > k + 1$ se cumple:

$$D(n) = Ack(n, n) > Ack(k + 1, n)$$

□

### Conclusión

Para que nuestro modelo de computación incluya todas las funciones "intuitivamente calculables" necesitamos poder utilizar bucles `While`.

## Conclusiones

Casi todas las funciones “usuales” son computables con bucles `For`.
Sólo hay que poder acotar el número de repeticiones de los bucles.

➔ Algunos programas no terminarán.

**Teorema (El problema de la parada, Turing, 1936)**

*No se puede determinar algorítmicamente si un programa va a terminar o no.*

**Proof.**

Por diagonalización. □

*¡Gracias por su atención!*