Recursión

Funciones recursivas

Jonatan Gómez Perdomo, Ph. D.
jgomezpe@unal.edu.co

Arles Rodríguez, Ph.D.
aerodriguezp@unal.edu.co

Camilo Cubides, Ph.D. (c)
eccubidesg@unal.edu.co

Carlos Andrés Sierra, M.Sc.
casierrav@unal.edu.co

Research Group on Artificial Life – Grupo de investigación en vida artificial – (Alife)
Computer and System Department
Engineering School
Universidad Nacional de Colombia

Agenda

Introducción I

Aunque a muchos el término *recursión* les parece extraño, la recursión es un concepto muy natural, tanto como para definir los números naturales.

Pensemos cuando tenemos un problema en nuestro diario vivir y un familiar muy respetado por nosotros dice "sea recursivo", ¿Qué nos quieren decir?

Que nos miremos a nosotros mismos, que sólo contemos con nuestros propios recursos para solucionar nuestros problema, que no se necesita algo adicional. Que pensemos que lo podemos solucionar para que lo solucionemos.

Agenda

Recursividad I

Dado que la mayoría de los lenguajes de programación disponen de un mecanismo para definir funciones (son estructurados), y dentro de la definición del cuerpo de las funciones se puede invocar el llamado a funciones (lo que se llama **composición de funciones** en matemáticas), una pregunta natural que uno se puede hacer es: ¿se puede invocar dentro de una función a la misma función para definirse?, en algunos lenguajes de programación esto si se puede hacer, a estos se les denominan lenguajes de programación con recursividad. Por ejemplo: C++, Java, Mathlab, Python, Lisp, y la mayoría de lenguajes de programación.

Recursividad II

El mecanismo de la recursión

Si un problema de cierto tamaño T puede ser solucionado usando instancias del mismo problema pero de menor tamaño t ($t < T$), y además se conoce la solución de algunas instancias de menor tamaño (t_0) que no dependan del problema, entonces se puede aplicar un mecanismo recursivo para implementar la solución del problema usando un lenguaje de programación.

Recursividad III

Definición (Definición débil de función recursiva)

Una función $f : A \to B$ se dice **recursiva** si y sólo si f está definida por casos (mediante un predicado sobre los argumentos), en donde al menos uno de los casos se define usando la misma función f y los argumentos, y al menos uno de los otros casos se define usando solamente los argumentos sin involucrar la función f.

Aquellos casos en los cuales se invoca a la misma función se llaman **casos recursivos**, y aquellos casos en los cuales no interviene la función se denominan **casos base**.

Nota

En teoría una función no necesita tener casos base, pero cualquier función recursiva escrita sin casos base generará un bucle infinito (ciclo sin fin).

Recursividad IV

Metodología para resolver problemas recursivamente

Solucionar problemas, y en particular programar, de manera "recursiva" se puede alcanzar siguiendo las tres máximas del profesor Jonatan Gómez.

1. Piense que está bien para que esté bien.
2. Siempre se es el primero y el último.
3. Un problema es simple o se puede dividir en problemas que se pueden solucionar siguiendo las tres máximas del profesor Jonatan Gómez.

Introducción

Problema 1.1

Problema

Calcular la suma de los números naturales desde el 0 hasta n.

Solución

Máximas dos y tres: El problema se puede solucionar si se pueden sumar los números naturales desde el 0 hasta en $n-1$ pues sólo sería sumar ese número con el número n.

Máxima uno: Se supone que la función que suma los números naturales desde el 0 hasta el n está bien por lo tanto se puede usar para sumar los números naturales desde el 0 hasta el $n-1$.

Máximas dos y tres: Sumar los números naturales desde el 0 hasta el 0 es fácil, se debe retornar 0 pues es el primer número natural. De esta manera será el último número en ser considerado en la suma.

Introducción

Problema 1.2

Solución (continuación)

$$sumar : \mathbb{N} \to \mathbb{N}$$

$$(n) \mapsto \begin{cases} n + sumar(n-1), & \text{si} \quad n > 0; \\ 0, & \text{en otro caso.} \end{cases}$$

El programa quedaría como sigue

```
def sumar(n):
    if n > 0:
        return n + sumar(n-1)
    else:
        return 0
```

Introducción

Problema 2.1

Problema

Calcular la suma de los números almacenados en una lista.

Solución

Máximas dos y tres: Supongamos que la lista tiene n números. El problema se puede solucionar si se pueden sumar los primeros $n-1$ números en la lista pues sólo sería sumar ese número con el número que está en la posición $n-1$.

Máxima uno: Se supone que la función que suma de los primeros n números de una lista está bien por lo tanto se puede usar para sumar los primeros $n-1$ números en la lista.

Máximas dos y tres: Sumar los primeros 0 números de la lista es fácil, se debe retornar 0 (no hay números que sumar).

Introducción

Problema 2.2

Solución (continuación)

$$sumar\_parcial : \mathbb{N}^* \times \mathbb{N} \to \mathbb{N}$$

$$(L, n) \mapsto \begin{cases} L_{n-1} + suma\_parcial(L, n-1), & \text{si} \quad n > 0; \\ 0, & \text{en otro caso.} \end{cases}$$

$$sumar\_lista : \mathbb{N}^m \to \mathbb{N}$$

$$(L) \mapsto sumar\_parcial\big(L, m\big)$$

Por notación, m es la longitud de la lista L.

Introducción

Problema 2.3

El programa quedaría:

```
def sumar_parcial(L,n):
    if n > 0:
        return L[n-1] + sumar_parcial(L,n-1)
    else:
        return 0

def sumar_lista(L):
    return sumar_parcial(L,len(L))
```

Introducción

Problema 3.1

Problema

Determinar si un carácter está en una cadena.

Solución

Máximas dos y tres: Supongamos que la cadena tiene n caracteres. El problema se puede solucionar si se puede determinar si un carácter está entre los primeros $n-1$ caracteres pues es mirar si es el último o está entre los primeros $n-1$ caracteres.

Máxima uno: Se supone que la función que determina si un carácter está entre los primeros n está bien, por lo tanto se puede usar para determinar si el carácter está entre los primeros $n-1$ caracteres.

Máximas dos y tres: Determinar si está entre los primeros 0 caracteres de la cadena es fácil, se debe retornar `falso`, ya que la cadena vacía no contiene caracteres.

Introducción

Problema 3.2

Solución (continuación)

$$buscar\_parcial : \mathbb{ASCII}^* \times \mathbb{ASCII} \times \mathbb{N} \to \mathbb{B}$$

$$(str, ch, n) \mapsto \begin{cases} (str_{n-1} \equiv ch) \vee buscar\_parcial(str, ch, n-1), & \text{si} \quad n > 0; \\ F, & \text{en otro caso.} \end{cases}$$

$$buscar : \mathbb{ASCII}^m \times \mathbb{ASCII} \to \mathbb{B}$$

$$(str, ch) \mapsto buscar\_parcial\big(str, ch, m\big)$$

Por notación, m es la longitud de la cadena str.

Introducción

Problema 3.3

El programa quedaría:

```
def buscar_parcial(str,ch,n):
    if n > 0:
        return (str[n-1] == ch) or buscar_parcial(str,ch,n-1)
    else:
        return False

def buscar(str,ch):
    return buscar_parcial(str,ch,len(str))
```


Agenda

Lectura y traducción I

Ejemplo

Sea f la función definida por:

$$f : \mathbb{N} \to \mathbb{B}$$

$$(n) \mapsto \begin{cases} V, & \text{si} \quad n = 0; \\ F, & \text{si} \quad n = 1; \\ f(n-2), & \text{en otro caso.} \end{cases}$$

- ¿Cuál es el resultado de evaluar la función f con los valores $n = 0, 1, 2, 3, 4, 5, 6, 7, 8$?
- ¿qué función matemática es f?
- ¿Cuál es su traducción a Python?

Lectura y traducción II

Ejemplo (continuación)

La traducción de la función f a Python es

```
def f(n):
    if n == 0:
        return True
    elif n == 1:
        return False
    else:
        return f(n-2)
```


Lectura y traducción III

Ejemplo

Sea g la función definida por:

$$g : \mathbb{N} \to \mathbb{N}$$

$$(n) \mapsto \begin{cases} 0, & \text{si} \quad n = 0; \\ 1, & \text{si} \quad n = 1; \\ 2, & \text{si} \quad n = 2; \\ g(n-3), & \text{en otro caso.} \end{cases}$$

- ¿Cuál es el resultado de evaluar la función g con los valores $n = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$?
- ¿qué función matemática es g?
- ¿Cuál es su traducción a Python?

Lectura y traducción IV

Ejemplo (continuación)

La traducción de la función g a Python es

```
def g(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    elif n == 2:
        return 2
    else:
        return g(n-3)
```

Lectura y traducción V

Ejemplo

Sea h la función definida por:

$$h : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

$$(n, m) \mapsto \begin{cases} n, & \text{si} \quad m = 0; \\ h(n+1, m-1), & \text{en otro caso.} \end{cases}$$

- ¿Cuál es el resultado de evaluar la función h con los valores $(n, m) = (2, 3), (8, 5), (6, 6)$?
- ¿qué función matemática es h?
- ¿Cuál es su traducción a Python?

Lectura y traducción VI

Ejemplo

La traducción de la función *h* a Python es

```
def h(n,m):
    if m == 0:
        return n
    else:
        return h(n+1,m-1)
```

Problema

Problema

Diseñe un modelo matemático que permita calcular la función **producto**, es decir, que reciba dos (2) números naturales y retorne la multiplicación del primer número por el segundo.

No se pueden utilizar los operadores de multiplicación ni de división (ni entera ni real).

Codifique el modelo matemático utilizando el lenguaje Python.

Agenda

Potencia de un número I

Ejemplo

En este ejemplo se definirá una función recursiva que permita hallar un número real elevado a un número natural. Para expresar una función que calcule esta operación, en primera instancia se construye la expresión *potencia* : $\mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$ que define la función que tiene como entrada un número real que representa la base y un número natural que indica el exponente, y como salida se obtendrá un número real que será la potencia. Por facilidad, aquí se asumirá que $0^0 = 1$.

Potencia de un número II

Ejemplo (continuación)

Ahora observesé que en general si se tiene una base b y un exponente n, entonces por definición

$$b^n = \underbrace{b * b * b * \cdots * b * b}_{n\text{–veces}}$$

si se usa la propiedad asociativa del producto de números reales, se tiene que

$$b^n = \underbrace{b * b * b * \cdots * b * b}_{n\text{–veces}} = \underbrace{(b * b * \cdots * b * b)}_{n-1\text{–veces}} * b$$

lo que es equivalente a

$$b^n = \underbrace{b * b * b * \cdots * b * b}_{n\text{–veces}} = \underbrace{(b * b * \cdots * b * b)}_{n-1\text{–veces}} * b = b^{n-1} * b$$

Potencia de un número III

A partir de esta observación se puede dar una definición recursiva usando funciones. La declaración de esta función junto con su cuerpo se hará de la siguiente manera

$$potencia(b, n) = p$$

Si se establecen las variables:

$$b := \text{Base}$$
$$n := \text{Exponente}$$
$$p := \text{Potencia } b^n$$

entonces

$$potencia : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$$
$$(b, n) \mapsto \begin{cases} 1, & \text{si} \quad n = 0; \\ potencia(b, n-1) * b, & \text{en otro caso.} \end{cases}$$

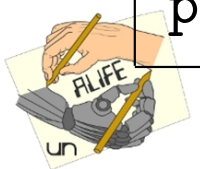
Potencia de un número IV

La codificación en Python de esta función es

```
def potencia(b, n):
    if n == 0:
        return 1
    else:
        return potencia(b,n-1) * b
```

Una solicitud por consola de la base y el exponente puede ser

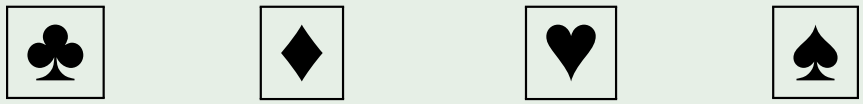
```
base = float(input("Por favor digite la base: "))
exp = int(input("Por favor digite el exponente: "))
print(base, "^", exp, "=", potencia(base, exp))
```



Principio del producto para conteo de listas I

Ejemplo

Suponga que se seleccionan cuatro cartas distintas de una baraja de póker, que se van a representar por los símbolos



si con estas cartas se conforma el conjunto $cartas = \{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$. ¿De cuántas formas distintas se pueden organizar las cartas?.

Solución

Como se van a listar todas las formas posibles en que se pueden organizar las cartas, el orden si importa.

Principio del producto para conteo de listas II

Solución (continuación)

Una estrategia para encontrar el número de listas puede ser el siguiente:

1. Se selecciona una carta del conjunto *cartas* de forma arbitraria pero fija, por ejemplo la carta ♦.
2. Ya fijada la carta ♦, el resto del trabajo consiste en hallar el número de formas distintas de organizar las cartas restantes, es decir, el conjunto $\textit{cartas} \setminus \{\diamondsuit\} = \{\clubsuit, \heartsuit, \spadesuit\}$.
3. Ahora por ejemplo se selecciona la carta ♠ de forma arbitraria pero fija.
4. A continuación, el trabajo se reduce a hallar el número de formas distintas de organizar las cartas restantes, es decir, el conjunto $\textit{cartas} \setminus \{\diamondsuit, \spadesuit\} = \{\clubsuit, \heartsuit\}$.

Principio del producto para conteo de listas III

Solución (continuación)

5. Posteriormente, por ejemplo se puede seleccionar de forma arbitraria pero fija la carta ♥.
6. Para finalizar, el trabajo se reduce a hallar el número de formas distintas de organizar las cartas restantes, es decir el conjunto $\mathit{cartas} \setminus \{♦, ♠, ♥\} = \{♣\}$. Como para este conjunto sólo se tiene una opción, entonces el número de formas distintas de organizar un conjunto de una carta es 1.

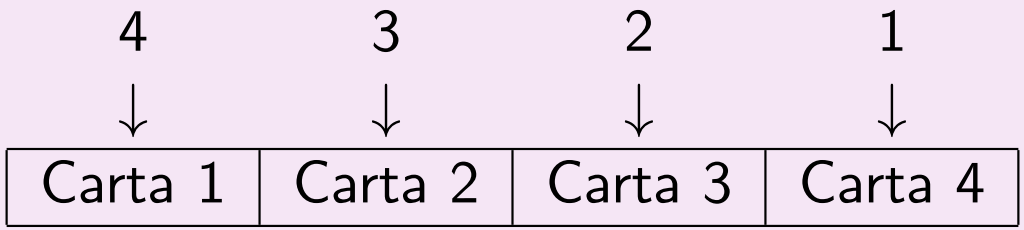
Siguiendo los pasos anteriores, se obtuvo la lista

♦ ♠ ♥ ♣

Principio del producto para conteo de listas IV

Solución (continuación)

Del análisis previo se puede concluir que el número de formas de listar los elementos de un conjunto con cuatro elementos se puede representar de la siguiente manera



de lo anterior y del principio del producto se puede afirmar que el número de todas la posibles listas que se forman con las cartas ♣, ♦, ♥, ♠ es

$$\prod_{i=1}^{4} i = 1 \cdot 2 \cdot 3 \cdot 4 = 24 = 4!$$

Función factorial I

En general, para un conjunto A con cardinal n ($|A| = n$), se tiene que el número de formas de listar todos los ordenamientos en que se pueden organizar los elementos de A es

$$\prod_{i=1}^{n} i = n * (n-1) * (n-2) * \cdots * 3 * 2 * 1$$

este valor que depende solamente de n, es una función, se denota por el símbolo $n!$ y se llama el factorial del número n. Para el caso del conjunto $\varnothing$, se puede demostrar que $0! = 1$. A partir de la asociatividad de la multiplicación de los naturales se tiene que

$$n! = n * \underbrace{(n-1) * (n-2) * \cdots * 3 * 2 * 1}_{n-1\text{–veces}} = n * (n-1)!$$

Función factorial II

A partir de lo anterior se puede obtener una función recursiva para el factorial de n ($n!$), que nombraremos

$$fact(n) = f$$

Si se establecen las variables:

$n :=$ Número al cual se le va a calcular el factorial

$f :=$ Factorial de n

entonces

Función factorial III

$$fact : \mathbb{N} \to \mathbb{N}$$

$$(n) \mapsto \begin{cases} 1, & \text{si} \quad n = 0; \\ n * fact(n-1), & \text{en otro caso.} \end{cases}$$

La codificación en Python de esta función es

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n-1)
```

Pago del interés compuesto mes vencido I

Ejemplo

Supónga que usted solicita un préstamo de $ 1'000.000 durante un año, el prestamista cobra un interés del 5% mensual con la modalidad de interés compuesto mes vencido. ¿Cuál es el total del dinero que debe pagar cuando ha transcurrido el año por el cual solicitó el préstamo, resolviendo el problema recursivamente?.

Solución

Para calcular el valor solicitado hay que observar que para cero (0) meses hay que pagar

$$pago(0) = \$1'000.000$$

Pago del interés compuesto mes vencido II

Solución (continuación)

Para calcular la cantidad de dinero que hay que pagar al cabo de los 12 meses, es suficiente con tener en cuenta que en el mes 12 se debe pagar lo que se debe en el mes 11 más los intereses que se producen en ese mes, por ejemplo si $pago(11)$ es el valor del pago en el mes 11, entonces la cantidad de dinero que hay que pagar en el mes 12 es igual a

$$\begin{aligned} pago(12) &= pago(11) + pago(11) * 0.05 \\ &= pago(11)(1 + 0.05) \\ &= pago(11) * 1.05 \end{aligned}$$

Pago del interés compuesto mes vencido III

Solución (continuación)

ahora para calcular el valor a pagar en el mes 11 se puede hacer un razonamiento similar, y para todos los meses anteriores hasta el mes cero (0) se hace de manera similar, lo que genera un esquema recursivo como el que se presenta a continuación

$$
\begin{aligned}
pago(12) &= pago(11) * 1.05 \\
pago(11) &= pago(10) * 1.05 \\
pago(10) &= pago(9) * 1.05 \\
pago(9) &= pago(8) * 1.05 \\
pago(8) &= pago(7) * 1.05 \\
pago(7) &= pago(6) * 1.05 \\
pago(6) &= pago(5) * 1.05
\end{aligned}
$$

$$
\begin{aligned}
pago(5) &= pago(4) * 1.05 \\
pago(4) &= pago(3) * 1.05 \\
pago(3) &= pago(2) * 1.05 \\
pago(2) &= pago(1) * 1.05 \\
pago(1) &= pago(0) * 1.05 \\
pago(0) &= \$\,1'000.000
\end{aligned}
$$

Pago del interés compuesto mes vencido IV

Solución (continuación)

Así, si $m = \$1'000.000$, $i = 0.05$ y $n = 12$, se tiene que el pago al cabo de los 12 meses será de $pago(12) = \$\,1'795.856,326$.

A partir de las observaciones anteriores ya se puede construir la regla recursiva con la que se puede calcular el interés compuesto mes vencido, y si se quiere una regla general que se pueda utilizar con cualquier monto m, con un interés i, y al cabo de n periodos se tiene la siguiente función recursiva de tres (3) parámetros

Pago del interés compuesto mes vencido V

Solución (continuación)

$$pago(m, i, n) = valor$$

donde se tienen las variables

$m :=$ Cantidad de dinero solicitado como prestamo

$i :=$ Interes

$n :=$ Número de meses por el cual se solicita el pretamo

$valor :=$ Valor total a pagar por el prestamo de la cantidad m por n meses con un interés i utilizando el método de interés compuesto mes vencido

entonces

Pago del interés compuesto mes vencido VI

Solución (continuación)

$$pago : \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{N} \to \mathbb{R}^+$$

$$(m, i, n) \mapsto \begin{cases} m, & n = 0; \\ pago(m, i, n-1) * (1+i), & \text{en otro caso.} \end{cases}$$

Pago del interés compuesto mes vencido VI

Solución (continuación)

La codificación en Python de esta función es

```
def pago(m, i, n):
    if n == 0:
        return m
    else:
        return pago(m, i, n-1) * (1+i)

print(pago(1E6, 0.05, 12))
```


Los conejos y los números de Fibonacci I

Ejemplo

Una pareja de conejos recién nacidos (uno de cada sexo) se liberan en una isla. Los conejos no pueden tener descendencia hasta que cumplen dos meses. Una vez que cumplen dos meses, cada pareja de conejos tiene como descendencia otra pareja de conejos cada mes[a]. ¿Cuál es la cantidad de parejas de conejos en la isla una vez transcurrido un año, suponiendo que ningún conejo muere?.

---

[a] Este problema fué propuesto originalmente por el italiano Leonardo Pisano Bigollo (1170–1250), más conocido como Leonardo de Pisa o Fibonacci (que significa hijo de Bonacci, *filius Bonacci*) en su libro *Liber abaci* publicado en 1202.

Los conejos y los números de Fibonacci II

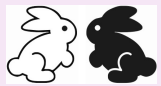
Solución

Si f_n denota la cantidad de parejas de conejos en el mes n, entonces, en el mes cero, en éste aun no se ha hecho la liberación de la pareja de conejos, por lo tanto la cantidad de parejas es $f_0 = 0$.

$$n = 0,\ f_0 = 0$$

Durante el primer mes, en este se hace la liberación de la primera pareja de conejos, pero aún no han alcanzado la edad para reproducirse, por lo tanto, no ha habido descendencia, por lo que $f_1 = 1$.

$$n = 1,\ f_1 = 1$$

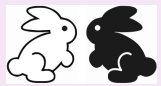


Los conejos y los números de Fibonacci III

Solución (continuación)

Durante el segundo mes, ya había una pareja de conejos del mes anterior y éstos aún no han alcanzado la edad para reproducirse, por lo tanto, no hubo descendencia, de donde f_2 es igual a la cantidad de conejos que habían en el mes anterior más la descendencia que produjeron las parejas de más de dos meses, es decir, $f_2 = 1$.

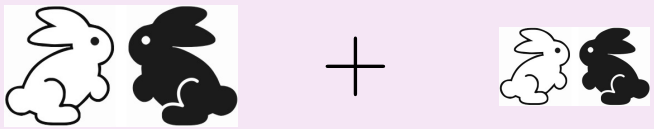
$$n = 2,\ f_2 = f_1 + f_0 = 1 + 0 = 1$$



Los conejos y los números de Fibonacci IV

Solución (continuación)

Durante el tercer mes, ya había una pareja de conejos del mes anterior y durante el transcurso de este mismo mes los conejos alcanzaron la madures para reproducirse, por lo tanto hubo descendencia, de donde f_3 es igual a la cantidad de conejos del mes anterior más la descendencia que se produjo en este mes, es decir, $f_3 = 2$.

$n = 3,\ f_3 = f_2 + f_1 = 1 + 1 = 2$ 

Los conejos y los números de Fibonacci V

Solución (continuación)

Durante el cuarto mes ya habían dos parejas de conejos del mes anterior, y la pareja madura es la que había en el segundo mes, por lo tanto, la descendencia fue generada sólo por esa pareja, de donde f_4 es igual a la cantidad de parejas del mes anterior más la descendencia que generé la pareja del segundo mes, es decir, $f_4 = f_3 + f_2 = 2 + 1 = 3$.

$n = 4,\ f_4 = f_3 + f_2 = 2 + 1 = 3$ 

Los conejos y los números de Fibonacci VI

Solución (continuación)

Durante el quinto mes ya habían tres parejas de conejos del mes anterior, y de éstas hay dos parejas maduras, que son las que habían en el tercer mes, por lo tanto, la descendencia fue generada por esas dos parejas, de donde f_5 es igual a la cantidad de parejas del mes anterior más la descendencia que generen las parejas del tercer mes, es decir, $f_5 = f_4 + f_3 = 3 + 2 = 5$.

$n = 5,\ f_5 = f_4 + f_3 = 3 + 2 = 5$

Los conejos y los números de Fibonacci VII

Solución (continuación)

Haciendo análisis similares se obtienen los siguientes resultados:

$$\begin{aligned}
&\text{Para } n = 6, \quad \text{se tiene que} \quad f_6 = f_5 + f_4 = 5 + 3 = 8 \\
&\text{Para } n = 7, \quad \text{se tiene que} \quad f_7 = f_6 + f_5 = 8 + 5 = 13 \\
&\text{Para } n = 8, \quad \text{se tiene que} \quad f_8 = f_7 + f_6 = 13 + 8 = 21 \\
&\text{Para } n = 9, \quad \text{se tiene que} \quad f_9 = f_8 + f_7 = 21 + 13 = 34 \\
&\text{Para } n = 10, \quad \text{se tiene que} \quad f_{10} = f_9 + f_8 = 34 + 21 = 55 \\
&\text{Para } n = 11, \quad \text{se tiene que} \quad f_{11} = f_{10} + f_9 = 55 + 34 = 89 \\
&\text{Para } n = 12, \quad \text{se tiene que} \quad f_{12} = f_{11} + f_{10} = 89 + 55 = 144
\end{aligned}$$

Los conejos y los números de Fibonacci VIII

Solución (continuación)

De aquí que, transcurrido el primer año, en la isla habrán 144 parejas de conejos.

A los números que son generados utilizando esta regla se les conoce como números de Fibonacci.

A partir del análisis anterior, se puede diseñar una función recursiva que permite calcular cualquier número de Fibonacci.

Los conejos y los números de Fibonacci IX

Solución (continuación)

$$fibo(n) = f$$

donde se tienen las variables

$n :=$ Número del cual se desea calcular su número de Fibonacci

$f :=$ Número de Fibonacci de n

entonces

$$fibo : \mathbb{N} \to \mathbb{N}$$

$$(n) \mapsto \begin{cases} 0, & \text{si} \quad n = 0; \\ 1, & \text{si} \quad n = 1; \\ fibo(n-1) + fibo(n-2), & \text{en otro caso.} \end{cases}$$

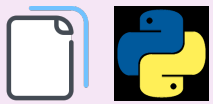
Los conejos y los números de Fibonacci X

Solución (continuación)

Una codificación en Python de esta función es

```
def fibo(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    return fibo(n - 1) + fibo(n - 2)

print(fibo(12))
```



Agenda

Teorema fundamental de la programación recursiva

Teorema (Teorema fundamental de la programación recursiva)

Un lenguaje de programación es completo en Turing si tiene valores enteros no negativos, funciones aritméticas elementales sobre dichos valores, así como un mecanismo para definir nuevas funciones utilizando las funciones ya existentes (composición), la selección (if) y la recursión.

Problemas III

Problemas

1. Modele mediante una función matemática y diseñe un programa recursivo que calcule la suma de los primeros n cuadrados de los números naturales $\left(\sum_{i=0}^{n} i^2\right)$.
2. Modele mediante una función matemática y diseñe un programa sin cadenas, tuplas o listas que retorne el último dígito de un número natural n (leído de izquierda a derecha). Por ejemplo, $\mathit{ultimo}(13579) = 9$.
3. Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que dado un número natural n elimine el último dígito del número. Por ejemplo, $\mathit{elimina\_ult}(654321) = 65432$.

Problemas IV

Problemas

4. Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que determine la cantidad de dígitos que componen un número natural n. Por ejemplo, $longitud(1230321) = 7$.
5. Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que calcule la suma de los dígitos que componen un número natural n. Por ejemplo, $suma\_digitos(123456) = 21$.

Problemas V

Problemas

6. Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que retorne el primer dígito de un número natural n (leído de izquierda a derecha). Por ejemplo, $primero(86420) = 8$.
7. Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que dado un número natural n elimine el primer dígito del número. Por ejemplo, $elimina\_pri(654321) = 54321$.
8. Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que dado un número natural n inserte un dígito al comienzo del número. Por ejemplo, $inserta(7, 654321) = 7654321$.

Problemas VI

Problemas

9. Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que invierta la cifras de un número n dado. Por ejemplo, $inversa(654321) = 123456$.
10. Modele mediante una función matemática y diseñe un programa recursivo sin cadenas, tuplas o listas que determine si un número es capicua. Un número se dice palíndromo si al leerlo de izquierda a derecha es lo mismo que leerlo de derecha a izquierda. Por ejemplo, $capicua(1) = V$, $capicua(1234321) = V$, $capicua(123421) = F$.