Caso de Estudio I: "Números Primos"

Esta es una actividad opcional.

En este caso de estudios,

- revisaremos el concepto de números primos,
- algoritmos asociados a ellos, e
- implementaremos funciones para algunos de esos algoritmos en un módulo de nombre **`primos.py`**.

El módulo `primos.py` parte inicialmente vacío:

```
# Modulo Primos
```

El caso de estudios consiste en una *actividad o desafío principal* y *actividades o desafíos complementarios*.

Actividad Principal

Para esta actividad, debe primero ver el siguiente video:

Identificando Números Primos (YouTube)

Una vez visto el video debe completar el desafío principal propuesto en el video.

Se recomienda escribir la función o funciones pedidas dentro del módulo de nombre `primos.py`.

La Solución está disponible más abajo en la sección soluciones. Se recomienda mirarlas sólo después de haber intentado resolver este problema por un rato.

Actividades Complementarias

Como actividad OPCIONAL le sugerimos resolver los siguientes problemas:

- **primos en un rango.** ej: 11 13 17 19 en 10-20. Su función debe retornar un string con la lista de los primos en el rango, por ejemplo:
  - primosEnRango(10,20) retorna "11 13 17 19"
- **primeros n primos.** ej: primeros 5: 2 3 5 7 11. Su función debe retornar un string con la lista de los primeros n primos, por ejemplo:
  - primerosPrimos(5) retorna "2 3 5 7 11 "

- **primo siguiente.** ej: de 23 es 29
- **i-ésimo primo.** ej: 1° es 2, 2° es 3, ...

Las soluciones están al final.

Otras actividades opcionales

Y para quienes aún tiene más ganas de investigar algoritmos con primos, están los siguientes desafíos. (Todo esto es OPCIONAL).

- **factores primos.** ej: 12 es 2 x 2 x 3 Dado un entero, esta función debe retornar un string con la lista de los divisores primos del entero. Por ejemplo,
  - factoresPrimos(12) retorna "2 2 3 "
- **primos relativos** (primos entre sí). ej: 9 y 4

Se recomienda poner sus funciones en el módulo `primos`.

No entregaremos soluciones para estos últimos.

Soluciones

+ Code + Text

Función "esPrimo"

- Indica si un numero **n** es primo (True) o no (False)
- Un número primo es aquel que tiene **dos** divisores exactos, **1** y **sí mismo**.
- **Ejemplo**: esPrimo(5) retorna True, esPrimo(9) retorna False
- Solución:
  - Enfoque de fuerza bruta (i.e., probar todos los números), tal como se describe en el video.

Se recomienda usar *parámetros por omisión* de Python en la función esPrimo. Si no sabe qué es eso, vea al final.

¿Cuál sería el algoritmo?

- ¿Qué significa buscar primos por fuerza bruta?
- ¿Cuál es el caso base? (recuerde que 0 y 1 no son primos)

Código:

```
# esPrimo: int -> bool
# indica si un numero es primo (True) o no (False)
```

```
# indica si un numero es primo (True) o no (False)
# Ejemplo: esPrimo(5) retorna True, esPrimo(9) retorna False
def esPrimo(n,divisor=2):

  assert type(n) == int and n>=2

  if n <= divisor:  #solo divide por si mismo, es primo
    return True

  if n % divisor == 0: #divide perfectamente es decir no es primo
    return False

  return esPrimo(n, divisor+1)



# test:
assert esPrimo(5)
assert not esPrimo(9)
```

Opcional

¿Podemos mejorar esto para que sea más eficiente?

Por ejemplo: podemos calcular solo en caso de n impar

- El caso en que n es par es fácil, qué debemos checkear para saber?
- El caso que no sea par es entonces el que debemos verificar:
    - No es necesario revisar todos los números impares como divisor.
    - Basta con llegar hasta $n < divisor^2$
    - Si n no es primo, entonces: $n = a \cdot b$
    - En ese caso, el valor máximo que puede tomar a ó b es $a = b$
    - $\Rightarrow n = a^2$

Código:

```
# esPrimoImpar: int -> bool
# version de esPrimo solo para n impares y
# se detiene cuando divisor es mayor a la raiz del numero
# Ejemplo: esPrimoImpar(13)-> True, esPrimoImpar(9)->False
def esPrimoImpar(n, divisor=3):

  assert type(n)==int and n>=3 and n%2==1 # nos aseguramos que es impar
```

```
  if divisor**2 > n :
    return True

  if n % divisor == 0:
    return False

  return esPrimoImpar(n,divisor+2)


assert not esPrimoImpar(9)
assert esPrimoImpar(13)
```

Luego podemos integrar todo en una sola función llamada `esPrimoOptimizado()` que sirve para pares e impares

```
# esPrimoOptimizado: int -> bool
# version optimizada de esPrimo que utiliza solo numeros impares y
# se detiene cuando divisor es mayor a la raiz del numero
# Ejemplo: esPrimoOptimizado(13)-> True, esPrimoOptimizado(14)-> False
def esPrimoOptimizado(n):

  assert type(n) == int and n>=2

  if n==2: # es 2
    return True

  if n%2==0:  # es par
    return False

  # es impar
  return esPrimoImpar(n)

assert not esPrimoOptimizado(12)
assert esPrimoOptimizado(13)
assert not esPrimoOptimizado(9)
```

También podemos juntar todo en una sola función:

```
# esPrimo: int -> bool
# version optimizada de esPrimo para pares e impares
# se detiene cuando divisor es mayor a la raiz del numero
# Ejemplo: esPrimo(13)-> True, esPrimo(14)-> False
def esPrimo(n,divisor=3):

  assert type(n) == int and n>=2
```

```
  if n==2:
    return True

  if n%2==0:  # es par
    return False

  if divisor**2 > n:
    return True

  if n % divisor == 0: # es divisible perfectamente por un número menor a n
    return False

  return esPrimo(n,divisor+2)

assert not esPrimo(12)
assert esPrimo(11)



# testPrimos: None -> None
# Lee numeros hasta que aparece un numero menor que 2
# e imprime si son primos o no
# ej: lee "17", "9", "7", "1" escribe "si", "no", "si"
def testPrimos():
    n = int(input("numero entero? "))
    if n<2:
        return # no escribimos nada mas
    if esPrimo(n):
        print("si ")
    else:
        print("no ")
    testPrimos()



testPrimos()
```

Escribir a pantalla los números primos dentro de cierto rango

```
# rango: int int -> str
# escribe primos entre x e y
# ej: rango(2,10) -> "2 3 5 7 " (con espacio al final)
def rango(x,y):

  assert type(x) == int and type(y)==int and x>=2

  if x>y:
    return ""  # retornamos string vacio
```

```
  if esPrimo(x):
    return str(x) + " " + rango(x+1,y)
  else:
    return rango(x+1,y)

# Tests
assert rango(2,10) == "2 3 5 7 "


rango(2,100)
```

Escribir los primeros n números primos

```
# primeros: int -> str
# escribe los n primeros numeros primos
# ej: primeros(5) -> "2 3 5 7 11 "

def primeros(n, indice=2, contador=1):

  if contador > n:
    return ""

  if esPrimo(indice):
    contador = contador+1
    return str(indice) + " " + primeros(n, indice+1, contador)
  else:
    return primeros(n, indice+1, contador)

# Tests
assert primeros(5) == "2 3 5 7 11 "


primeros(10)
```

Escribir el siguiente número primo de un entero n

```
# siguiente: int->int
# primo siguiente a un entero dado
# ej: siguiente(2)->3, siguiente(13)->17
def siguiente(n):
  assert type(n)==int and n>=1

  if n<=2:
    return n+1

  if esPrimo(n+1):
    return n+1
```

```
  return siguiente(n+1)

# test:
assert siguiente(2)==3
assert siguiente(13)==17



siguiente(200)
```

Escribir el i_ésimo número primo

```
#i_esimo: int -> int
#i_esimo numero primo
#ejs: i_esimo(1)->2, i_esimo(2)->3
def i_esimo(i,indice=1,primo=2):

  assert type(i)==int and i>=1

  if indice==i:
    return primo

  return i_esimo(i,indice+1,siguiente(primo))

# test:
assert i_esimo(1)==2
assert i_esimo(5)==11



i_esimo(4)
```

Parámetros por omisión en Python

Es una técnica de Python que permite darle un valor **inicial** a un parámetro *en el caso que dicho parámetro NO sea dado al llamar la función*. Por ejemplo, si queremos poder llamar a una función con dos parámetros `f(x,y)` en dos casos distintos:

- `f(10,20)` : caso normal, el segundo parámetro es 20.
- `f(10)` : nos gustaría poder llamarlo así, sin poner el segundo parámetro, y que en ese caso el segundo parámetro tenga un valor fijo, por ej. 20. O sea, que dicho llamado sea equivalente a `f(10,20)`.

Ejemplo de uso de parámetros por omisión

Supongamos que queremos saber cuál es el mayor dígito en un entero N. Por ejemplo, mayorDigito(752381081) debe retornar 8.

```
def mayorDigito(N): ...
```

Una manera posible de resolverla es recursivamente, donde en cada llamada recursiva sacamos el último dígito y lo comparamos con el mayor dígito encontrado, y seguimos recursivamente. Para que esta estrategia funcione, tenemos que ir "recordando" cuál es el mayor dígito encontrado en la llamada. Una posibilidad es pasarle este mayor dígito como parámetro de la función recursiva.

Entonces, el problema es que para que nuestro algoritmo funcione, debemos pasarle a la función `mayorDigito(N)` un segundo parámetro, el valor del **mayor dígito visto hasta el momento en la recursión**. O sea, debiera llamarse más bien:

```
def mayorDigito(N,m): ...
```

Y en m irá el mayor dígito visto. Pero esto **no luce muy bonito**, pues la primera vez que llamemos a la función debemos pasarle un 0 como parámetro `m`. Por ejemplo

```
# Ejemplo de uso
e = int(input("Ingrese un número entero de varios dígitos?"))
mayord = mayorDigito(e,0)
print("El mayor dígito en el número es ",mayord)
```

Para evitar poner "0" como parámetro, se usan la notación de parámetro por omisión. Esto consiste

```
# Contrato
# mayorDigito: int -> int
# calcula el mayor digito que tiene un entero
# Ej: mayorDigito(1234321) debe retornar 4
def mayorDigito(n,m=0):
  if n==0:
    # Caso base: se acabo n, el mayor dígito es m
    return m

  ultimoDigito = n % 10
  nSinUltimoDigito = n // 10
  # comparamos el ultimo digito con el mayor digito hasta el momento (el cual recordar
  if ultimoDigito>m: # es mayor!
    return mayorDigito(nSinUltimoDigito, ultimoDigito)
  else:  # no es mayor
    return mayorDigito(nSinUltimoDigito,m)

# Tests
assert mayorDigito(752381081)==8
assert mayorDigito(1234565321)==6
```

```
# Ejemplo de uso (programa principal)
e = int(input("Ingrese un número entero de varios dígitos?"))

# Ojo que no necesitamos darle segundo parámetro a la función (supone m=0)
mayord = mayorDigito(e)
print("El mayor dígito en el número es ",mayord)
```

Es necesario usar parámetros por omisión de Python?: En realidad, no.

Siempre es posible usar una función distinta, no recursiva primero, y que esa función llame a la función recursiva con los dos parámetros.

Por ejemplo:

```
def obtieneMayorDigito(n):
  return mayorDigito(n,0)
```

Con eso, la función `mayorDigito(n,m)` puede escribirse normalmente, con dos parámetros, esto es, tal como está más arriba, pero sin usar m=0 en el encabezado.