5 Récursivité

July 26, 2024

1 Opérations arithmétiques

1.1 PGCD

[1]:
```
def pgcd(n:int, m:int)->int:
    if n == 0:
        return m
    elif m < n:
        return pgcd(m, n)
    else:
        return pgcd(n, m%n)
```

[2]:
```
pgcd(42, 36)
```

[2]: 6

1.2 Multiplication

[3]:
```
def mult(a:int, b:int)->int:
    if b == 0:
        return 0
    else:
        return a + mult(a, b-1)
```

[4]:
```
mult(3, 6)
```

[4]: 18

1.3 Puissance

[5]:
```
def puiss(a:int, b:int)->int:
    if b == 0:
        return 1
    else:
        return a * puiss(a, b-1)
```

[6]:
```
puiss(2, 3)
```

[6]: 8

1.4 Division euclidienne

[7]:
```
def reste(a:int, b:int)->int:
    if a < b:
        return a
    else:
        return reste(a-b, b)
```

[8]:
```
reste(10, 3)
```

[8]: 1

[9]:
```
def quotient(a:int, b:int)->int:
    if a < b:
        return 0
    else:
        return 1 + quotient(a-b, b)
```

[10]:
```
quotient(10, 3)
```

[10]: 3

1.5 Exponentiation rapide

[11]:
```
def exp(a:int, n:int)->int:
    if n == 0:
        return 1
    else:
        e = exp(a, n//2)
        if n%2==1:
            return a * e * e
        else:
            return e * e
```

[12]:
```
exp(2, 8)
```

[12]: 256

2 Fonctions récursives sur les listes

3 len et replicate

[13]:
```
def len(li:[int])->int:
    if li == []:
        return 0
    else:
        return 1 + len(li[1:])
```

[14]:
```
len([1, 2, 3])
```

[14]: 3

[15]:
```
def replicate(el:int, n:int)->[int]:
    if n == 0:
        return []
    else:
        return [el] + replicate(el, n-1)
```

[16]:
```
replicate(0, 3)
```

[16]: [0, 0, 0]

4 Sum et product

[17]:
```
def sum(li:[int])->int:
    if li == []:
        return 0
    else:
        return li[0] + sum(li[1:])
```

[18]:
```
sum([1, 2, 3])
```

[18]: 6

[19]:
```
def product(li:[int])->int:
    if li == []:
        return 1
    else:
        return li[0] * product(li[1:])
```

[20]:
```
product([1, 2, 3, 4])
```

[20]: 24

5 Reverse

[21]:
```
def reverse(li:[int])->[int]:
    if li == []:
        return []
    else:
        return reverse(li[1:]) + [li[0]]
```

[22]:
```
reverse([1, 2, 3])
```

[22]:
```
[3, 2, 1]
```

6 Zip

[23]:
```
def zip(s1:[int], s2:[int])->[(int, int)]:
    if s1 == [] or s2 == []:
        return []
    else:
        return [(s1[0], s2[0])] + zip(s1[1:], s2[1:])
```

[24]:
```
zip([1, 2, 3], [1, 2, 3, 4])
```

[24]:
```
[(1, 1), (2, 2), (3, 3)]
```

7 Map

[25]:
```
def map(f:callable, li:list)->list:
    if li == []:
        return []
    else:
        return [f(li[0])] + map(f, li[1:])
```

[26]:
```
def inc(x:int)->int:
    return x+1
```

[27]:
```
map(inc, [1, 2, 3])
```

[27]:
```
[2, 3, 4]
```

8 Filter

[28]:
```
def filter(cond:callable, li:list)->list:
    if li == []:
        return []
    else:
        if cond(li[0]):
```

```
            return [li[0]] + filter(cond, li[1:])
        else:
            return filter(cond, li[1:])
```

[29]:
```
def estPositif(x:int)->bool:
    return x >= 0
```

[30]:
```
filter(estPositif, [1, -2, 3, -4])
```

[30]: [1, 3]

9 Max et mem

[31]:
```
def max(li:[int])->int:
    if len(li) == 1:
        return li[0]
    else:
        M = max(li[1:])
        if li[0] < M:
            return M
        else:
            return li[0]
```

[32]:
```
max([6, 3, 4, 7, 2, 1])
```

[32]: 7

[1]:
```
def mem(li:[int], val:[int])->bool:
    if li == []:
        return False
    else:
        return (val == li[0]) or mem(li[1:], val)
```

[2]:
```
mem([1, 2, 3], 2)
```

[2]: True

10 Merge

[35]:
```
def merge(li1:list, li2:list)->list:
    if li1 == []:
        return li2
    elif li2 == []:
        return li1
    else:
        if li1[0] < li2[0]:
```

```
            return [li1[0]] + merge(li1[1:], li2)
        else:
            return [li2[0]] + merge(li1, li2[1:])
```

[36]:
```
merge([0, 2, 4], [1, 3, 5, 7])
```

[36]:
```
[0, 1, 2, 3, 4, 5, 7]
```

11 Horner

[37]:
```
def P(a:int, x:float)->float:
    if len(a) == 0:
        return 0
    else:
        return a[0] + x*P(a[1:], x)
```

[38]:
```
a = [2, 3, 6]
x = 7
P(a, x)
```

[38]:
```
317
```

12 Recherche séquentielle

[39]:
```
def seq_search(li:[int], el:int, i=0)->int:
    if li == []:
        return None
    else:
        if el == li[0]:
            return i
        else:
            return seq_search(li[1:], el, i+1)
```

[40]:
```
print(seq_search([1, 2, 3, 4], 7))
```

```
None
```

13 Recherche dichotomique

[41]:
```
def dich_search(li:[int], el:int, i=0)->int:
    if li == []:
        return None
    else:
        im = len(li)//2
        if li[im] > el:
            return dich_search(li[:im], el, i)
```

```
        elif li[im] < el:
            return dich_search(li[im+1:], el, i+im+1)
        else:
            return im + i
```

[42]:
```
print(dich_search([1, 3, 4], 5))
```

```
None
```

14 All et any

[43]:
```
def all(cond:callable, li:list)->bool:
    if li == []:
        return True
    else:
        return cond(li[0]) and all(cond, li[1:])
```

[44]:
```
def estPair(n:int)->bool:
    return n%2 == 0
```

[45]:
```
all(estPair, [0, 2, 3, 4, 5, 7, 8])
```

[45]: False

[46]:
```
def any(cond:callable, li:list)->bool:
    if li == []:
        return False
    else:
        return cond(li[0]) or any(cond, li[1:])
```

[47]:
```
any(estPair, [0, 2, 3, 4, 5, 7, 8])
```

[47]: True

15 Suite de Fibonacci

[48]:
```
def F(n:int)->int:
    if n == 0 or n == 1:
        return n
    else:
        return F(n-1) + F(n-2)
```

[49]:
```
for i in range(8):
    print(F(i))
```

```
0
1
```

```
1
2
3
5
8
13
```

[50]:
```
def F2(n:int)->int:
    F0 = 0
    F1 = 1
    if n == 0:
        return F0
    elif n == 1:
        return F1
    else:
        for i in range(2, n+1):
            F1, F0 = F0+F1, F1
    return F1
```

[51]:
```
for i in range(8):
    print(F2(i))
```

```
0
1
1
2
3
5
8
13
```

La fonction `F(n)` est très inefficace, dans la mesure où elle recalcule beaucoup de fois certains termes (par exemple, pour calculer `F(4)` on calcule deux fois `F(2)`).

16 Numération binaire

[52]:
```
def bin(n:int)->[int]:
    if n == 0:
        return []
    else:
        return [n%2] + bin(n//2)
```

[53]:
```
bin(13)
```

[53]:
```
[1, 0, 1, 1]
```

[54]:
```
def ch_bin(n:int, ch='')->None:
    if n == 0:
```

```
        print(ch)
    else:
        ch_bin(n-1, ch + '0')
        ch_bin(n-1, ch + '1')
```

[55]:
```
ch_bin(3)
```

```
000
001
010
011
100
101
110
111
```

17 Dénombrement

[56]:
```
def ins(el:int, li:[int])->[[int]]:
    if li == []:
        return [[el]]
    else:
        return [[el] + li] + [ [li[0]] + i for i in ins(el, li[1:])]
```

[57]:
```
ins(4, [1, 2, 3])
```

[57]: `[[4, 1, 2, 3], [1, 4, 2, 3], [1, 2, 4, 3], [1, 2, 3, 4]]`

[58]:
```
def perms(li:[int])->[[int]]:
    if li == []:
        return [[]]
    else:
        return [p for el in perms(li[1:]) for p in ins(li[0], el)]
```

[59]:
```
perms([1, 2, 3])
```

[59]: `[[1, 2, 3], [2, 1, 3], [2, 3, 1], [1, 3, 2], [3, 1, 2], [3, 2, 1]]`

[60]:
```
def subs(li:[int])->[[int]]:
    if li == []:
        return [[]]
    else:
        return [[li[0]] + el for el in subs(li[1:])] + subs(li[1:])
```

[61]:
```
subs([1, 2, 3])
```

[61]: `[[1, 2, 3], [1, 2], [1, 3], [1], [2, 3], [2], [3], []]`

[62]:
```
def choices(li:[int])->[[int]]:
    return [p for s in subs(li) for p in perms(s)]
```

[63]:
```
choices([1, 2, 3])
```

[63]:
```
[[1, 2, 3], [2, 1, 3], [2, 3, 1], [1, 3, 2], [3, 1, 2], [3, 2, 1], [1, 2], [2,
1], [1, 3], [3, 1], [1], [2, 3], [3, 2], [2], [3], []]
```

[ ]:
```

```