

Unit 18.

ການນຳໃຊ້ຝັງຊັມ Recursion

ឧបតម្លៃសិក្សាអនុវត្តន៍

- ✓ ដើរតាមរយៈការបង្កើតនូវការបញ្ចូលrecursive.
- ✓ ដើរតាមរយៈការបង្កើតនូវការបញ្ចូល factorial និងសារធានាបុរាណ factorial ដោយការស្វែងរកទីផ្សារ។
- ✓ សារធានាបុរាណ factorial និង Fibonacci ដោយការបង្កើតនូវការបញ្ចូល recursive និងការបង្កើតនូវការបញ្ចូល Fibonacci ដោយការស្វែងរកទីផ្សារ។
- ✓ ដើរតាមរយៈការបង្កើតនូវការបញ្ចូលrecursive និងសារធានាបុរាណ Fibonacci ដោយការស្វែងរកទីផ្សារ។

ພາບລວມຂອງການຮຽນຮູ້

- ✓ ຮຽນຮູ້ວິທີແກ່ໃຂບັນຫາດ້ວຍການດ້ວຍການເອັນໃຊ້ຝັງຊັນເອງ.
- ✓ ຮຽນຮູ້ວິທີນຳເອົາ factorial ໄປໃຊ້ໃນຄໍາສັ່ງ loop ເພື່ອກວດສອບປະສິດທິພາບຂອງຝັງຊັນ recursive.
- ✓ ຮຽນຮູ້ວິທີສ້າງຝັງຊັນ recursive ແລະ ເອັນໃຊ້ຝັງຊັນດັ່ງກ່າວເພື່ອຄົ້ນຫາຕົວເລກ factorial ແລະ Fibonacci.
- ✓ ຮຽນຮູ້ຂໍເສຍ ແລະ ຂໍ້ຈໍາກັດຂອງການເອັນໃຊ້ recursive ແລະ ວິທີການນຳເອົາລຳດັບ Fibonacci ໄປໃຊ້ໃນຫາງທີ່ດີຂຶ້ນຜ່ານການຈົດຈໍາ.

ສິ່ງຈໍາເປັນຕ້ອງຮັບຈາກ Units ຜ່ານມາ

- ✓ ວິທີກຳນົດ ແລະ ເອັນໃຊ້ຝັງຊັນທີ່ຜູ້ໃຊ້ກຳນົດຂຶ້ນ ໂດຍໃຊ້ຄໍາສັ່ງ def
- ✓ ວິທີສິ່ງຄ່າໄປທີ່ຝັງຊັນຢ່າງມີປະສິດທິພາບ ໂດຍໃຊ້ variable parameters, default parameters ແລະ keyword parameters.
- ✓ ວິທີຄືນຄ່າຫຼາຍຄ່າໂດຍໃຊ້ຄໍາສັ່ງ return ຂອງຝັງຊັນ.

Keywords

Recursive Call

Factorial

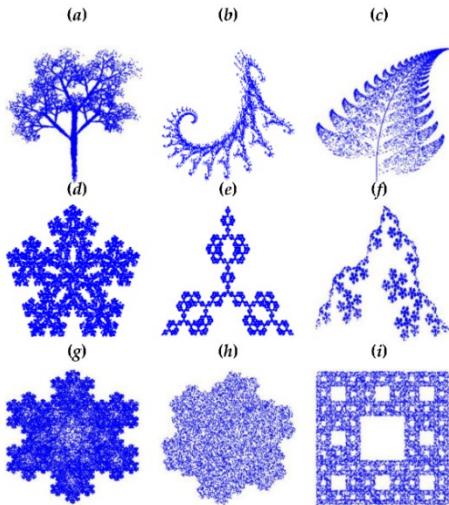
Fibonacci Sequence

memoization

Mission

1. បំណងទីតាំងនៃការប្រើប្រាស់ការពិភាក្សាប្រចាំថ្ងៃ

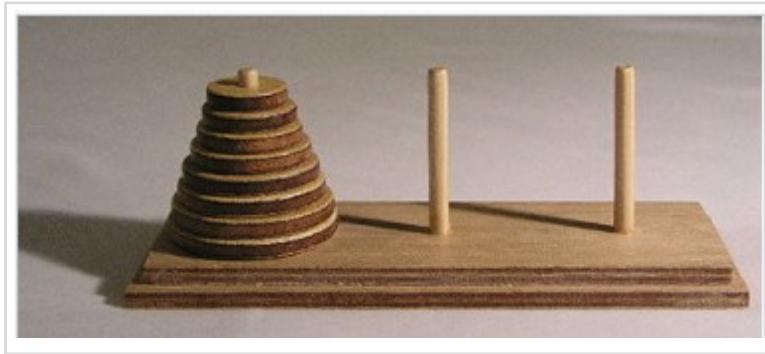
1.1. រួចរាល់ពីការប្រើប្រាស់ការពិភាក្សាប្រចាំថ្ងៃ នៅក្នុងការប្រើប្រាស់ការពិភាក្សាប្រចាំថ្ងៃ



- មានអ្នកដែលបានប្រើប្រាស់ការពិភាក្សាប្រចាំថ្ងៃដើម្បីបង្ហាញនូវការរួចរាល់នៃការប្រើប្រាស់ការពិភាក្សាប្រចាំថ្ងៃ។
- យើងអាចបង្ហាញនូវការរួចរាល់នៃការប្រើប្រាស់ការពិភាក្សាប្រចាំថ្ងៃដោយចូលទៅកាន់សាខាដំឡើងនៃការប្រើប្រាស់ការពិភាក្សាប្រចាំថ្ងៃ។
- នូវការរួចរាល់នៃការប្រើប្រាស់ការពិភាក្សាប្រចាំថ្ងៃ នឹងធ្វើឡើងជាផ្លូវការប្រចាំថ្ងៃ។ ការប្រើប្រាស់ការពិភាក្សាប្រចាំថ្ងៃ នឹងធ្វើឡើងជាផ្លូវការប្រចាំថ្ងៃ។
- ការប្រើប្រាស់ការពិភាក្សាប្រចាំថ្ងៃ នឹងធ្វើឡើងជាផ្លូវការប្រចាំថ្ងៃ។

1. ບັນຫາຂອງໂລກແຫ່ງຄວາມເປັນຈິງ

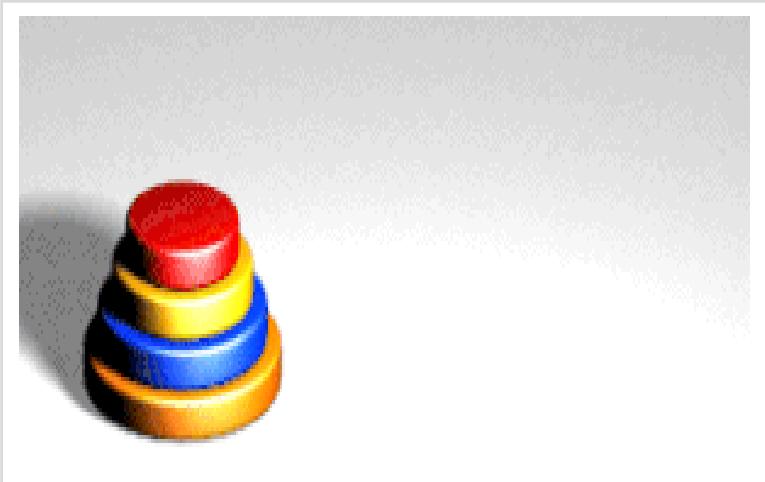
1.2. ບັນຫາທຳຄອຍ Hanoi



- ▶ ບັນຫາທຳຄອຍ Hanoi ເປັນຕົວແນະຂອງບັນຫາທີ່ຂໍອນຂ້າງງ່າຍໃນການແກ້ໄຂບັນຫາ ໂດຍນຳໃຊ້ວິທີການວິນຊ້າ.
- ▶ ທຳຄອຍ Hanoi ແມ່ນເກມທີ່ເປັນຄ້າຍຄືແຜ່ນ disk ທີ່ສະແດງໃນຮູບດ້ານຊ້າຍ ທີ່ຖືກຍ້າຍ ໄປຕາມແທ່ງອື່ນໆຕາມກົດທີ່ກໍານົດໄວ້.
- ▶ ເວລາຫຼຸ້ນເກມນີ້ ຈະມີຈຳນວນຄັ້ງໃນການຍ້າຍທີ່ເໜນຈະສົມທີ່ສຸດ. ຖ້າມີ n disk ເຮົາສາມາດ ຍ້າຍ disk ຫັງໝົດໄປໄດ້ $2^n - 1$ ຄັ້ງ.
- ▶ ຖ້າບໍ່ຮູ້ຂັ້ນຕອນການຍ້າຍ, ເຮົາອາດຈະຍ້າຍຫຼາຍກວ່າຈຳນວນຄັ້ງທີ່ເໜນຈະສົມ.
- ▶ ມາຮຽນຮູ້ຂັ້ນຕອນການສ້າງໂປຣແກຣມທຳຄອຍ Hanoi.

2. ការແກ້ໄຂបັນຫາ

2.1. ໄຂປິດສະໜາທຳຄອຍ Hanoi



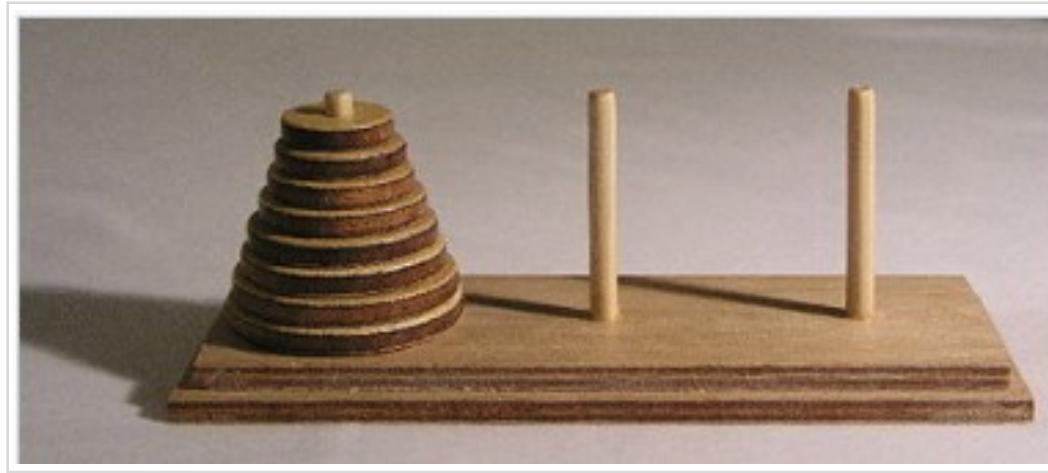
https://en.wikipedia.org/wiki/Tower_of_Hanoi

- ▶ ທຳຄອຍ Hanoi ແມ່ນເກມປິດສະໜາປະເພດໜຶ່ງ (puzzle game).
- ▶ ມີເທິງ ແລະ ແຜ່ນກົມສາມອັນທີ່ມີຂະໜາດຕ່າງກັນ ທີ່ສາມາດໃສ່ລົງໄປເສົາເລົ່ານີ້ໄດ້ຝຶດືກີ, ກ່ອນຈະເລີ່ມໄຂປິດສະໜາ, ແຜນຈະລຽງຊ້ອນກັນຈາກລຸ່ມຂຶ້ນທີ່ໂດຍໃຫ້ແຜນນ້ອຍປອງຢູ່ທີ່ແຜ່ນໃຫຍ່ຕາມລຳດັບ.
- ▶ ຈຸດປະສົງຂອງເກມແມ່ນຍ້າຍແຜນຢູ່ໃນເທິງໜຶ່ງໄປອີກເທິງໜຶ່ງຕາມລຳດັບ ແລະ ສ້າງມັນຂຶ້ນມາໃໝ່ໂດຍປະຕິບັດຕາມເງື່ອນໄຂສໍ່ປະການດັ່ງຕໍ່ໄປນີ້.
- ▶ ມີ 4 ເງື່ອນໄຂຂອງເກມຄື:
 1. ຍ້າຍ disk ຈຳນວນ n disk ທີ່ມີຂະໜາດຕ່າງກັນຈາກຈຸດເລີ່ມຕົ້ນ A ໄປຫາຈຸດສຸດທ້າຍ C.
 2. ສາມາດຍ້າຍແຜນ Disk ໄດ້ຄັ້ງລະໜຶ່ງແຜນເທິງໜຶ່ນ.
 3. ການຍ້າຍ disk, ແມ່ນໃຫ້ເອົາແຜ່ນທີ່ຢູ່ດ້ານເທິງສຸດຂອງເທິງ ແລ້ວຍ້າຍໄປໃສ່ດ້ານເທິງສຸດຂອງເທິງອື່ນ.
 4. ໃນຂະບວນການຍ້າຍແຜ່ນ disk, ບໍ່ສາມາດວາງແຜ່ນໃຫຍ່ໃສ່ເທິງແຜ່ນນ້ອຍ.

2. ការងារខ្លួន

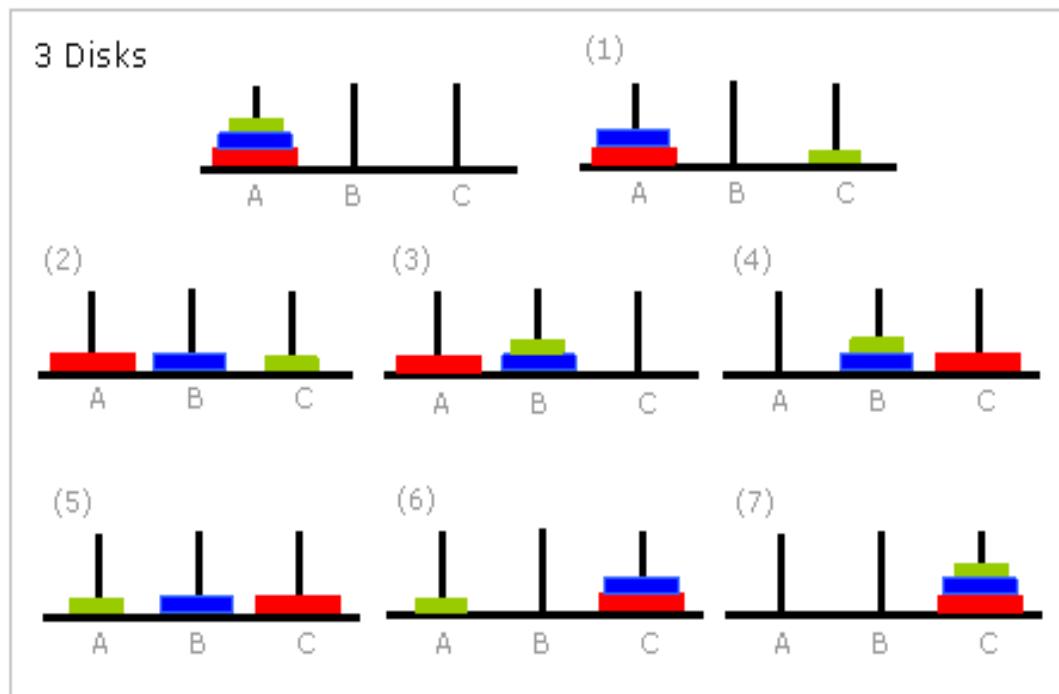
2.1. ខ្លឹមពិភោគនាប៊ូតុលាមទី Hanoi

- | សាមាត្រូវបានផ្តល់ព័ត៌មានរបៀបចាប់ដំឡើងលាមទី Hanoi, ដោយចូលទៅកាន់ https://en.wikipedia.org/wiki/Tower_of_Hanoi.
- | នឹងរាយការណ៍ប៊ូតុលាមទី Hanoi ដោយប្រើប្រាស់ការបញ្ជីការងារខ្លួន។



3. Mission

3.1. ขั้นตอนวิธีของงานแก้ไขปิดสะทมห์ดอย Hanoi



- ▶ ถ้ามีแผนกรูว กำسامาดယายແຜນ disk ຈາກແກນເລີ່ມຕົ້ນໄປຫາແກນສຸດທ້າຍໄດ້ຄັງດຽວເທົ່ານັ້ນ.
- ▶ ສິມມຸດມີ disk ຫັງໝົດ 3 ແຜນ ແລະ ມີແກນ 3 ແກນ.
 1. ເພື່ອຍ້າຍ disk 3 ຕໍາທີ່ສຸດ ໄປແກນຂາເຊົ້າ ໃຫ້ຍ້າຍ 3-1 discs ໄປໄວ້ແກນສືມ.
 2. ຍ້າຍແຜນ Disc 3 ໄປແກນຂາເຊົ້າ.
 3. ຍ້າຍ 3-1 discs ຈາກແກນສືມໄປແກນຂາເຊົ້າ.
- ▶ ເຮົາສາມາດແກ້ໄຂບັນຫານີ້ດ້ວຍວິທີການເອັ້ນຊັ້າ.

https://en.wikipedia.org/wiki/Tower_of_Hanoi

3. Mission

3.2. ដំឡើង Hanoi()

```
1 if n == 1:  
2     print('Number {}, {} -> {}'.format(n, from_, to_))  
3 else:  
4     hanoi(n-1, from_, via_, to_)  
5     print('Number {}, {} -> {}'.format(n, from_, to_))  
6     hanoi(n-1, via_, to_, from_)
```

- ▶ ឯកដំឡើង, តាមតីវិធីណែនាំនៃការប្រើប្រាស់ការពារមូលដ្ឋាន disk.
- ▶ នឹងរាយការណ៍ការប្រើប្រាស់ការដំឡើង Hanoi. និមួយនាមដំឡើង Hanoi, និងរាយការណ៍ការប្រើប្រាស់ការដំឡើង Hanoi នូវការប្រើប្រាស់ការពារមូលដ្ឋាន។

```
1 hanoi(3, 'A', 'B', 'C')
```

3. Mission

3.3. ນີ້ແມ່ນການເຮັດວຽກຂອງທຳ Hanoi. (if n = 3)

```
1 def hanoi(n, from_, to_, via_):
2     if n == 1:
3         print('Number {}, {} -> {}'.format(n, from_, to_))
4     else:
5         hanoi(n-1, from_, via_, to_)
6         print('Number {}, {} -> {}'.format(n, from_, to_))
7         hanoi(n-1, via_, to_, from_)
8 hanoi(3, 'A', 'C', 'B')
```

Number 1, A -> C
Number 2, A -> B
Number 1, C -> B
Number 3, A -> C
Number 1, B -> A
Number 2, B -> C
Number 1, A -> C

3. Mission

3.4. ချို့ယော်ပြန္တော်မ

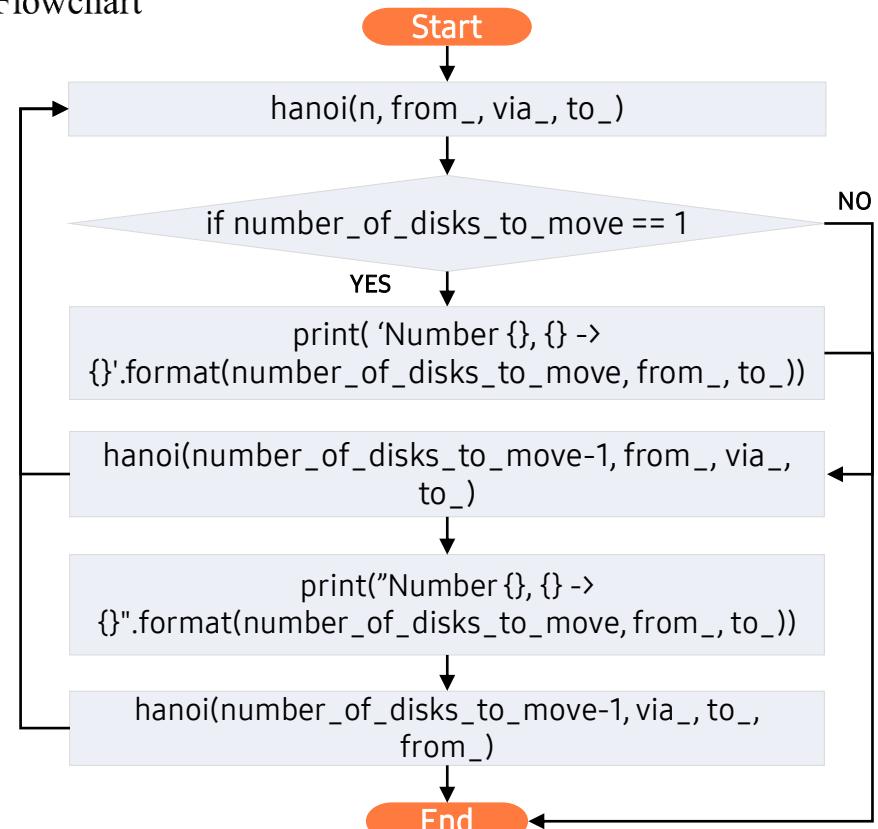
| Pseudocode

```

[1] Start
[2] Call the function hanoi(n, from_, to_, via_)
[3] if there is only one disk then {
    [4] Move the disk from from_ to to_
[5] else { Move the (n-1) disks from from_ to to_
    Call the function hanoi(n-1, from_, via_, to_)
    [6] Move the largest disk n from_ to_ destination
    [7] Move (n-1) disks in via_ to the destination
        from_ is used as an auxiliary rod
        Call the function hanoi(n-1, via_, to_, from_) }
[8] End

```

| Flowchart



3. Mission

3.5. ໂຄດຄໍາສັງສຸດທ້າຍຂອງ ຫໍາ Hanoi

```
1 def hanoi(n, from_, to_, via_):
2     if n == 1:
3         print('Number {}, {} -> {}'.format(n, from_, to_))
4     else:
5         hanoi(n-1, from_, via_, to_)
6         print('Number {}, {} -> {}'.format(n, from_, to_))
7         hanoi(n-1, via_, to_, from_)
8 hanoi(3, 'A', 'C', 'B')
```

| Key concept

1. ການຊອກຫາ Factorial ດ້ວຍຄໍາສັ່ງ Loop

1.1. ການຊອກຫາ Factorial

| Factorial ເປັນຄໍາທີ່ມາຈາກຄະນິດສາດ ແລະ ຂຽນເປັນ $n!$ ເຊິ່ງ n ເປັນຈຳນວນທຳມະຊາດ.

- ▶ factorial ຫຼື $n!$ ແມ່ນຜົນຄຸນຂອງຈຳນວນທຳມະຊາດຈາກ 1 ເຖິງ n .

$$n! = \prod_{k=1}^n k = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 3 \cdot 2 \cdot 1$$

1. ການຊອກຫາ Factorial ດ້ວຍຄໍາສັ່ງ Loop

1.2. Factorial ແລະ for loop

| ກໍານົດ ແລະ ນຳໃຊ້ factorial ດ້ວຍໂຄດຄໍາສັ່ງຢ່າງ ໂດຍປັດສະຈາກຄໍາສັ່ງ return. for loop ສາມາດນຳໃຊ້ດັ່ງລຸ່ມນີ້.

```
1 n = int(input('Enter a number : '))
2 fact = 1
3 for i in range(1, n+1):
4     fact = fact * i
5
6 print('{}! = {}'.format(n, fact))
```

Enter a number : 5

5! = 120

Line 3, 4

- ຈາກ 1 ເຖິງ n, ຕົວປ່ຽນ fact ແມ່ນຈະຖືກຄຸນດ້ວຍ i ແລ້ວເກັບໄວ້ໃນຕົວປ່ຽນ fact ຄືເກົ່າໄປເລື້ອຍໆ .

2. ការគិតមាត្រា Factorial ដោយផ្ទាយខ្លួនខ្លឹមខ្លា

2.1. ផ្ទាយខ្លួនខ្លាំបែងផ្ទាយខ្លួនរបៀបណា?

- | ផ្ទាយខ្លួនខ្លឹមខ្លាំ បែងផ្ទាយខ្លួនទីផ្សារម៉ោង, មួយបែងចែកជាការរាយការណ៍ដែលមិនមែនការសម្រាប់បង្ហាញឡើយ និង សាមសារអាជីវកម្ម។
- | នីមួយៗនៃ Factorial និងការគិតមាត្រា : $n! = n * (n - 1)!$, if $n \leq 1$: return 1

```
def factorial(5):
    if n <= 1 :
        return 1
    else :
        return n * factorial(n-1)
```

↳ 5 * factorial(4)

return 1 : ការបញ្ចប់នៃការសម្រាប់បង្ហាញនៅក្នុងការគិតមាត្រា

↳ 4 * factorial(3)

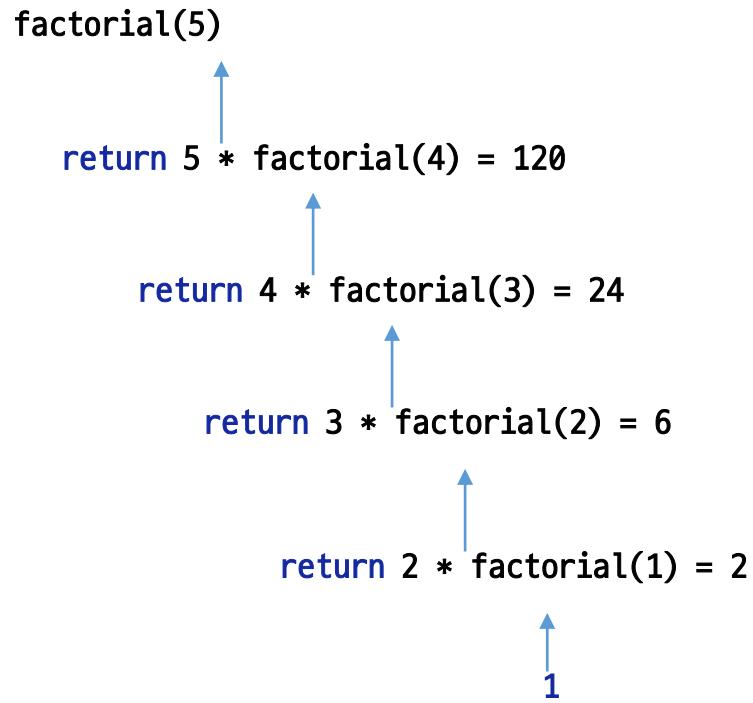
↳ 3 * factorial(2)

↳ 2 * factorial(1)

2. ការគិតមាត្រា Factorial ដោយផ្ទាយខ្លួនខ្លឹមខ្លា

2.2. ការថ្លាខទេរងផ្ទាយខ្លួនខ្លឹមខ្លា

| ការថ្លាខទេរង factorial ផ្ទាយខ្លួនខ្លឹមខ្លា។ Factorial(1) returns 1. ដោយ factorial(2), $2 * \text{factorial}(1) = 2 * 1$ ແມ່ນស្ថិតិភាព។ សូមបានរាយការណ៍ដែលបានរាយការណ៍ដោយការគិតមាត្រា។



2. ការគិតមាត្រា Factorial ដោយឃើញខ្លួនខ្លឹមខ្លា

2.2. ការថ្លាខទេរឃើញខ្លួនខ្លឹមខ្លា

 Focus ឲ្យឃើញខ្លួនខ្លា, ពីរូនិងផ្សេងៗមិតាមសំណើនេះបានយ៉ាងចិត្ត។

| ដោយពីរូនិង ១! = 1, n! = n*(n-1)! តាមដែនឹងនេះ ត្រូវគិតមាត្រាដោយប្រើប្រាស់ការតាមដុំលើការគិតមាត្រាដែលបានរាយការណ៍។

```
1 def factorial(n):      # Implement recursive function of n!
2     if n <= 1 :          # Termination condition is required
3         return 1
4     else :
5         return n * factorial(n-1)    # n * (n-1)! implementation by definition
6
7 n = 5
8 print('{}! = {}'.format(n, factorial(n)))
```

5! = 120

| ចុចសំណើនេះ ឲ្យឃើញខ្លួនខ្លា និងស្ថាបនិតាបានរាយការណ៍។

| តាម ១! = 1, ស្ថាបនិតាបាន តាមការតាមដុំលើការគិតមាត្រាដែលបានរាយការណ៍។

ອີກຂັ້ນຕອນໜຶ່ງ

1. ຂັ້ນຕອນວິທີຂອງການເຄື່ອນຊັ້ນ ແລະ Divide-and-Conquer

- | ในການເອັນຊ້າ, ບັນຫາກຳຄົງ ເມື່ອຟັງຂັນເອັນຕົວມັນເອງ ແລະ ຂະໜາດ ແມ່ນ n , ມັນເປັນເລື່ອງປຶກກະທິທີ່ຈະເອັນມັນດ້ວຍການແບ່ງບັນຫາອອກເປັນບັນຫານີ້ອຍກວ່າ n .
 - | ວິທີແກ້ບັນຫານີ້ກຳຄົນມີໃຊ້ຂັ້ນຕອນວິທີ divide-and-conquer.
 - | ຂັ້ນຕອນວິທີ divide-and-conquer ສາມາດເອັນອີກຢ່າງໜຶ່ງວ່າການເອັນຊ້າ multi-branch. ຕົວຢ່າງ factorial ທີ່ສະແດງໄວ້ກ່ອນໜ້ານີ້ບໍ່ແມ່ນຂັ້ນຕອນວິທີ divide-and-conquer, ເພະວ່າ ມັນເປັນການເອັນຊ້າແບບ one-branch ເຊິ່ງການເອັນຊ້າຈະຮັດພຽງຄັ້ງດຽວ
 - | ເມື່ອຂະໜາດຂອງບັນຫາແມ່ນ n , ໃຫ້ແບ່ງບັນຫາອອກເປັນສອງສ່ວນດ້ວຍຂະໜາດ $n/2$ ແລະ ແຕ່ລະບັນຫາຜ່ານ argument ໄປເອັນຊ້າ, ເຊິ່ງຈະເປັນຂັ້ນຕອນວິທີ divide-and-conquer, ນອກຈາກການແບ່ງເຄິ່ງແລ້ວ ວິທີການເອັນຊ້າ ດ້ວຍການແບ່ງຫຼາຍກວ່າສອງບັນຫາກໍລົວແຕ່ປັນ ຂັ້ນຕອນວິທີ divide-and-conquer.
 - ▶ ໃນບັນຫາ divide-and-conque, ມີຂໍ້ຈໍາກັດວ່າ ຜົນລວມຂອງບັນຫາທີ່ຖືກແບ່ງນັ້ນຕ້ອງບໍ່ຫຼາຍກວ່າບັນຫາເດີມ.
 - | ທ້າຮຽນຮູ້ໂຄງສ້າງຂໍ້ມູນ ຫຼື algorithms ມາກ່ອນ, ຈະຮັດໃຫ້ຮຽນວິທີການລຽງລຳດັບໄດ້ໄວ ເຊິ່ງເອັນວ່າ quick sort.
 - ▶ sort algorithm ສາມາດນຳໄປໃຊ້ໃນວິທີການ divide-and-conquer ໂດຍນຳໃຊ້ການເອັນຊ້າ.

3. ຂໍບົກຜ່ອງຂອງຝັງຊັນເອີ້ນຊ້າ ແລະ ການຈິດຈໍາ

3.1. ຂໍດີ ແລະ ຂໍບົກຜ່ອງຝັງຊັນເອີ້ນຊ້າ ແລະ ຄຳສັ່ງ loop

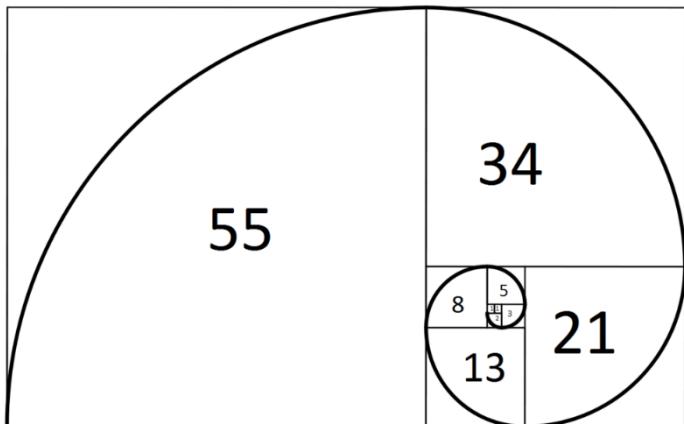
| ສົມທຽບລະຫວ່າງຂຶ້ດີ (pros) ແລະ ຂໍເສັຍ (cons) ຂອງຝັງຊັນເອີ້ນຊ້າ ຕ່າງສະແດງໃນຕາຕະລາງລຸ່ມນີ້.

ຝັງຊັນເອີ້ນຊ້າ (Recursive Functions)		ຄຳສັ່ງ Loop (Loop Statements)
Pros	ໂຄດຄຳສັ່ງບໍ່ຂັບຂ້ອນ (Simple code)	ການປະມວນຜົນໄວ (Fast)
Cons	ໃຊ້ໜ່ວຍຄວາມຈໍາຫຼາຍ (Use a lot of memory) ການປະມວນຜົນຊ້າ (Slow)	ໂຄດຄຳສັ່ງຂັບຂ້ອນ (Complex code)

3. ຂໍ້ມູນກ່ຽວຂ້ອງເລກີ່າ ແລະ ການຈິດຈຳ

3.2. ການກຳນົດລຳດັບຂອງ Fibonacci

| ລຳດັບຂອງ Fibonacci ແມ່ນລຳດັບ term ທີ່ 1 ແລະ term ທີ່ 2 ເປັນ 1, ແລະ term ທີ່ຕາມມາທັງໝົດແມ່ນຜົນລວມຂອງສອງເທິມກ່ອນໜ້ານີ້



ຮບແບບເລຂາຄະນິດທີ່ສ້າງດ້ວຍລຳດັບ Fibonacci

- term ເທິມທີ 1 F_0 ຂອງລຳດັບ Fibonacci ແມ່ນ 0, ສ່ວນ F_1 ແມ່ນ 1, ແລະ F_n ແມ່ນ ສະແດງດັ່ງລຸ່ມນີ້.

$$F_n = F_{n-1} + F_{n-2}$$

- ອີງຕາມການກຳນົດ, ລຳດັບຂອງ Fibonacci ແມ່ນ:
 - 0, 1, 1, 2, 3, 5, 8, 13, 21, ...
- ລຳດັບຂອງ Fibonacci ສາມາດສ້າງຮູບແບບເລຂາຄະນິດ ດັ່ງສະແດງໃນຮູບດ້ານຊ້າຍນີ້.

<https://shoark7.github.io/assets/img/algoritm/fibonacci-logo.png>

3. ខ្លឹមរងទែនខ្សោយដំឡើង និងការគិតថា

3.2. ការកំណើនលាតបខ្សោយ Fibonacci

| ការកំណើនលាតប Fibonacci ត្រូវបានធ្វើដោយប្រើប្រាស់ដំឡើងខ្សោយ។

- ▶ ប្រើប្រាស់ក្រុមសាស្ត្រក្នុងខ្សោយដំឡើងខ្សោយ Fibonacci សរុបតួនាទី។

```
1 def fibonacci(n):                      # A recursive implementation of the Fibonacci function
2     if n <= 1:                          # Termination condition of the Fibonacci function
3         return n
4     else:
5         return(fibonacci(n-1) + fibonacci(n-2)) # F_n = F_(n-1) + F_(n-2)
6
7 nterms = int(input("How many Fibonacci numbers do you want?"))
8
9 # Cannot find Fibonacci number if it is negative
10 if nterms <= 0:
11     print("Error : Enter a positive number.")
12 else:
13     print("Fibonacci sequence: ", end=' ')
14     for i in range(nterms):
15         print(fibonacci(i), end=' ')
```

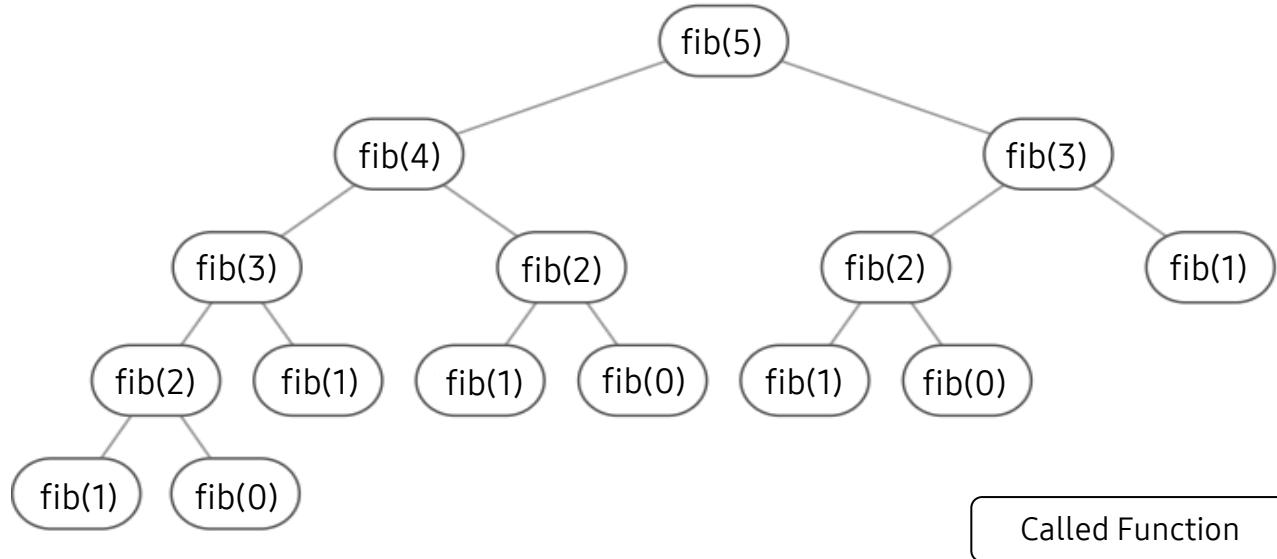
How many Fibonacci numbers do you want? 5

Fibonacci sequence: 0 1 1 2 3

3. ຂໍ້ມູນຜ່ອງຂອງຝັງຊັນເອັນຊຳ ແລະ ການຈົດຈໍາ

3.3. ການປະມວນຜົນຂອງຝັງຊັນ Fibonacci

- | ຮຽນຮູ້ຂະບວນການດໍາເນີນການຂອງຝັງຊັນ Fibonacci: $\text{fib}(5)$.
- | $\text{fib}(5)$ ເອັນ $\text{fib}(4)$ ແລະ $\text{fib}(3)$, $\text{fib}(4)$ ເອັນຊຳໆ $\text{fib}(3)$ ແລະ $\text{fib}(2)$. ຜົນໄດ້ຮັບແມ່ນມີໂຄງສ້າງຕົ້ນໄມ້ຂະໜາດໃຫຍ່.



3. ຂໍ້ມູນຜ່ອງຂອງຝັງຊັນເອັນຊຳ ແລະ ການຈິດຈຳ

3.4. ການນຳໄປໃຊ້ຂອງຝັງຊັນ Fibonacci

| ການໃຊ້ຂອງຝັງຊັນເອັນຊຳຂອງລຳດັບ Fibonacci ແມ່ນດັ່ງສະແດງລຸ່ມນີ້.

```
1 def fibonacci(n):
2     if n == 0:
3         return 0
4     elif n == 1:
5         return 1
6     else:
7         return fibonacci(n-1) + fibonacci(n-2)
8
9
10 print(fibonacci(10))
```

55

Line 7

- ຄ່າຈະເພີ່ມເມື່ອມີການເອັນຊຳຈົນກວ່າລົບຕາມເງື່ອນໄຂ ຈຶ່ງຢຸດ.

3. ຂໍ້ມູນຜ່ອງຂອງຝັງຊັນເອັນຊຳ ແລະ ການຈົດຈໍາ

3.5. ປະສິດທິພາບຂອງການນຳໃຊ້ຝັງຊັນ Fibonacci ເຊິ່ງເປັນຝັງຊັນເອັນຊຳ

- | ບັນຫາໃຫຍ່ຂອງຝັງຊັນເອັນຊຳກຳຕືີ ເປັນຝັງຊັນບໍ່ມີປະສິດທິພາບ ແພະວ່າ ມັນມີການຄໍານວນຊຳ ເຖິງແມ່ນວ່າ ຈະເປັນຄ່າທີ່ຄໍານວນໄປແລ້ວກຳຕາມ. ມາເບິ່ງຂະບວນການຫາລຳດັບ Fibonacci ທີ 30 ໃນຕົວຢ່າງລຸ່ມນີ້.
- | ພວກເຮົາຈະໃຊ້ໄມດຸນເພື່ອວັດແທກເວລາ. Timer() ເປັນຝັງຊັນທີ່ຢູ່ໃນໄມດຸນວັດແທກ, ຈະວັດແທກຄວາມໄວຂອງການດຳເນີນການຂອງຄໍາສັ່ງທີ່ບ້ອນເປັນ argument ທຳເລີດ. ສໍາລັບລາຍລະອຽດໃນເລື່ອງນີ້ໃຫ້ເບິ່ງໃນເວັບໄຊນີ້. (<https://docs.python.org/3/library/timeit.html>)

```
1 from timeit import *
2
3 def fibonacci_1(n):
4     if n == 0:
5         return 0
6     elif n == 1:
7         return 1
8     else:
9         return fibonacci_1(n-1) + fibonacci_1(n-2)
10
11 t3 = Timer("fibonacci_1(30)", "from __main__ import fibonacci_1")
12 print("fibonacci_1(30) * 20 times : ", t3.timeit(number = 20), "seconds")
```

fibonacci_1(30) * 20 times : 9.716442599999937 seconds

- ▶ ມັນຄົ້ນຫາລຳດັບທີ່ 30 ຈໍານວນ 20 ຄັ້ງ ແລະ ໃຊ້ເວລາ 9.7 ອິນາທີ. (t3.timeit(number = 20) repeats 20 times)
- ▶ ຖ້າມີການຄົ້ນຫາລຳດັບທີ່ສູງກວ່ານີ້ ມັນກໍຈະໃຊ້ເວລາຫຼາຍກວ່ານີ້.

3. ຂໍ້ມູນຜ່ອງຂອງຝັງຊັນເອັນຊຳ ແລະ ການຈົດຈຳ

3.6. ຝັງຊັນ Fibonacci ແລະ ການຈົດຈຳ

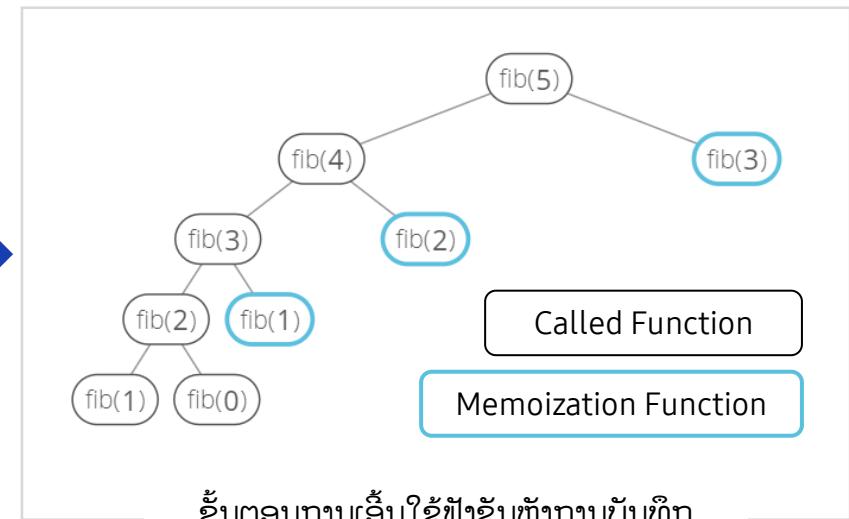
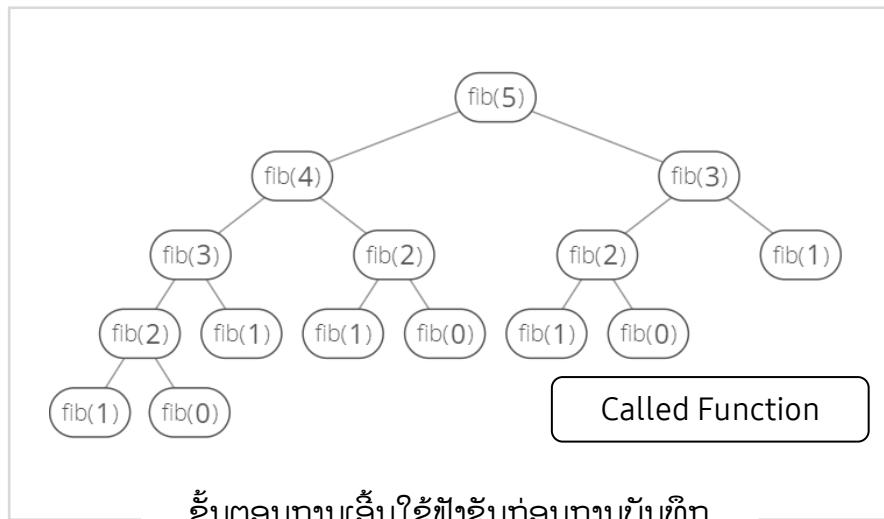
 **Focus** ການຈົດຈຳແມ່ນວິທີການເກັບຜົນຮັບຂອງບັນຫານ້ອຍໆທີ່ເກີດຂຶ້ນຊຳໄປເຊົາມາຫຼາຍໆຮັງ ແຕ່ຜົນຮັບຈະບໍ່ມີການປ່ຽນແປງ.

- | ເພື່ອແກ້ໄຂບັນຫາຂ້າງເທິງ, ຄວນໃຊ້ການຈົດຈຳ (**memorization**) ໃນ **Dynamic Programming (DP)**.
- | ການຂຽນໂປຣແກຣມແບບໄດ້ນາມິກ (Dynamic programming) ເປັນໜຶ່ງໃນຂັ້ນຕອນວິທີ ທີ່ໃຊ້ໃນການແກ້ໄຂບັນຫາ.
 - | ຈຸດສຳຄັນ :
 - ▶ ໃນການຂຽນໂປຣແກຣມແບບໄດ້ນາມິກ ແມ່ນຕ້ອງແກ້ໄຂບັນຫານ້ອຍໆເພື່ອນໍາໄປສູ່ແກ້ໄຂບັນຫາໃຫຍ່.
 - ▶ ຜົນໄດ້ຮັບຂອງບັນຫານ້ອຍໆມີກາຈະຄົງທີ່.
 - ▶ ຜົນຮັບຈາກບັນຫານ້ອຍໆ ຈະບໍ່ປ່ຽນແປງຜົນຮັບທີ່ເກີດຂຶ້ນຊຳໆ.
 - | ເມື່ອຜົນຮັບໄດ້ບັນທຶກຜ່ານການຈົດຈຳທີ່ເກີດຂຶ້ນແບບຊຳ, ການຄໍານວນທີ່ບໍ່ຈໍາເປັນສາມາດຂ້າມໄປໄດ້ ແລະ ສາມາດເອັນສະເພາະຄ່າທີ່ບັນທຶກໄວ້ໄດ້ຢ່າງວ່ອງໄວ.
 - | ຝັງຊັນເອັນຊຳ ຍັງຕ້ອງແກ້ໄຂບັນຫານ້ອຍໆໃຫ້ຄືບຕາມເງື່ອນໄຂຂອງບັນຫາໃຫຍ່ ແລະ ອາດຈະມີບັນຫານ້ອຍຂຶ້ນອີກໃນລະຫວ່າງການປະມວນຜົນ.

3. ຂໍ້ມູນຜ່ອງຂອງຝັງຊັນເອັນຊຳ ແລະ ການຈົດຈໍາ

3.6. ຝັງຊັນ Fibonacci ແລະ ການຈົດຈໍາ

- | ໃນຕົວຢ່າງນີ້ ແມ່ນການນຳໃຊ້ຝັງຊັນການຈົດຈໍາ ເຊັ່ນ: `fib(5)` ແມ່ນການຄົ້ນຫາໜາຍແລກ Fibonacci 5.
- | ຮູບດ້ານຊ້າຍ ເປັນການເອັນໃຊ້ຝັງຊັນ 15 ພັງຊັນ ໂດຍບໍ່ມີການບັນທຶກ. ສ່ວນຮູບດ້ານຂວາ ຈຳນວນການເອັນໃຊ້ຝັງຊັນຫຼຸດລົງເຫຼືອ 9 ຜ່ານການຈົດຈໍາ ຫຼື ບັນທຶກ. ການຈົດຈໍາສາມາດຮັດໄດ້ໂດຍຜ່ານ Python dictionary.



3. ຂໍ້ມູນຜ່ອງຂອງຝັງຊັນເອັນຊຳ ແລະ ການຈົດຈໍາ

3.7. ຝັງຊັນ Fibonacci ແລະ ຫຼັກການຂອງການຈົດຈໍາ

 Focus ເພື່ອປ້ອງກັນການນັບຊຳ, ຖ້າຄ່າຂອງເງື່ອນໄຂຖືກຈົດເວັບເປັນອີງປະກອບໃນ dic, ຄ່າທີ່ຄໍານວນບໍ່ຈໍາເປັນຕ້ອງຄໍານວນໃໝ່.

| ໂດດຄໍສິ່ງລຸ່ມນີ້ ແມ່ນການຄົ້ນຫາໜາຍເລກ Fibonacci ທີ n ດັ່ງສະແດງລຸ່ມນີ້.

```

1 from timeit import * # Import all classes and methods of timeit
2
3 dic = {0:0, 1:1} ←
4 def fibonacci_2(n):
5     if n in dic:
6         return dic[n] ←
7
8     dic[n] = fibonacci_2(n-1) + fibonacci_2(n-2) ←
9     return dic[n]
10
11 t2 = Timer("fibonacci_2(30)", "from __main__ import fibonacci_2")
12 print("fibonacci_2(30) * 20 times ", t2.timeit(number = 20), "seconds")

```

fibonacci_2(30) * 20 times 3.32999981360743e-05 seconds

| ລະຫັດທີ່ບໍ່ໄດ້ມີການຈົດຈໍາ ຫຼື ບໍ່ບັນທຶກ ຈະໃຊ້ເວລາ 9.7 ວິນາທີ, ສ່ວນລະຫັດທີ່ມີການຈົດຈໍາ ຫຼື ບັນທຶກ ຈະໃຊ້ເວລາ 0.000033 ວິນາທີ.

| ຖ້າປະສິດທິພາບຂອງຝັງຊັນເອັນຊຳ ຫຼຸດລົງ, ຈະເປັນການດີທີ່ຈະເພີ່ມປະສິດທິພາບ ໂດຍການຈົດເວັບຜົນຮັບທຸກຄັ້ງທີ່ມີການເອັນໃຊ້ຝັງຊັນແບບເອັນຊຳ

ປະກາດ dictionary ສໍາລັບການຈົດຈໍາ ດ້ວຍຕົວປ່ຽນ
global, dic = {0:0, 1:1} ເອັນໃຊ້ fibonacci_2(1)
returns 1, ເຊິ່ງແມ່ນ dic[1]

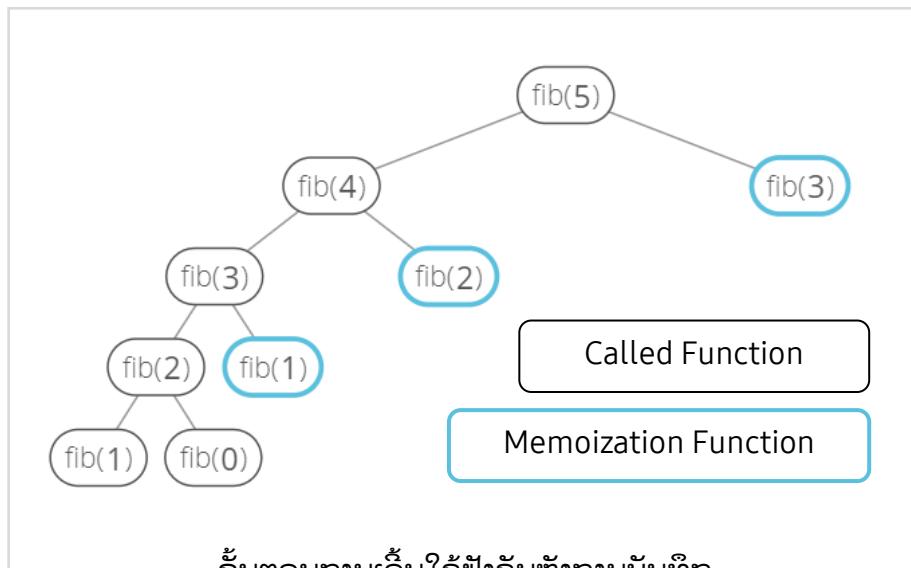
ຖ້າ dictionary ມີຜົນຮັບຈາກການຄໍານວນແລ້ວ, ມັນຈະສິ່ງຜົນຮັບຄືນ
ແລະ ບໍ່ມີການເອັນຊຳ.

ຖ້າ dictionary ບໍ່ມີຜົນຮັບຈາກການຄໍານວນ, ຝັງຊັນຈະທຳການເອັນຊຳ. ແລະ
ຜົນຂອງການເອັນນີ້ຈະຖືກເກັບໄວ້ໃນ dictionary.

3. ຂໍ້ມູນຜ່ອງຂອງຝັງຊັນເອັນຊຳ ແລະ ການຈົດຈໍາ

3.7. ຝັງຊັນ Fibonacci ແລະ ຫຼັກການຂອງການຈົດຈໍາ

 Focus ຮຽນຮູ້ວິທີອັບແດດ dictionaries.



The initial dictionary state : `dic = {0:0, 1:1}`



Call `fib(2)` : return `fib(1) + fib(0)`

Call of `fib(2)` : return `dic[0] + dic[1]`, which becomes 1
`dic = {0:0, 1:1, 2:1}` state updated



Call `fib(3)` : return `fib(2) + fib(1)`

Call `fib(3)` : return `dic[1] + dic[2]`, which becomes 2
`dic = {0:0, 1:1, 2:1, 3:2}` state updated



...

Paper coding

- ຕ້ອງເຂົ້າໃຈແນວຄິດພື້ນຖານຂອງຫຼັກສູດນີ້ໃຫ້ຄົບຖ້ວນກ່ອນຈະໄປສູ່ຂັ້ນຕອນຕໍ່ໄປ.
- ການທີ່ບໍ່ເຂົ້າໃຈແນວຄິດພື້ນຖານ ຈະເພີ່ມພາລະໃນການຮຽນຮູ້ຂອງຫຼັກສູດນີ້ ແລະ ຈະຮັດໃຫ້ບໍ່ປະສົບຜົນສໍາເລັດ.
- ມັນອາດຈະເປັນເລື່ອງທີ່ຍາກຕອນນີ້, ແຕ່ຖ້າຢາກປະສົບຜົນສໍາເລັດໄດ້ນັ້ນ ພວກເຮົາຂໍແນະນຳໃຫ້ເຂົ້າໃຈແນວຄິດພື້ນຖານ ນີ້ຢ່າງເລີກເຊິ່ງ ແລະ ກ້າວໄປສູ່ຂັ້ນຕອນຕໍ່ໄປ.

Q1. បែងពិវាលេក n និង តើម្មានជិតបរភោគចាប់ពី 1 ដល់ n. ចំណាំខ្លួនឯងខ្លួនដែលបានផ្តល់ឱ្យខ្លួនបង្កើតឡើង (recursive function).

Condition for Execution	Enter a number : 10 55
Time	5 minutes



Write the entire code and the expected output results in the note.

Q2.

ໃນ Python ມີຕົວດໍາເນີນການ ** ທີ່ລະບຸເປັນສື່ແຈຈະຕຸລັດ. ຢ່າງໃດກໍຕາມ, ລອງໃຊ້ x ແລະ n ເປັນຕົວ input ໂດຍບໍ່ໃຊ້ຕົວດໍາເນີນການຂອງ Python ແລະ ນຳໃຊ້ຝຶ່ງຊັ້ນແບບເອັນເຂົ້າ (recursive function) ເພື່ອສະແດງຜົນຮັບ x^n . ຈຶ່ງຂຽນໂຄດຄໍາສັ່ງເພື່ອໃຫ້ໄດ້ຜົນຮັບຂອງ 2^{10} ເຊິ່ງໃນນີ້ x=2, n=10. ດັ່ງສະແດງລຸ່ມນີ້

Condition for Execution

```
Enter x : 2  
Enter n : 10  
1024
```

Time

5 minutes



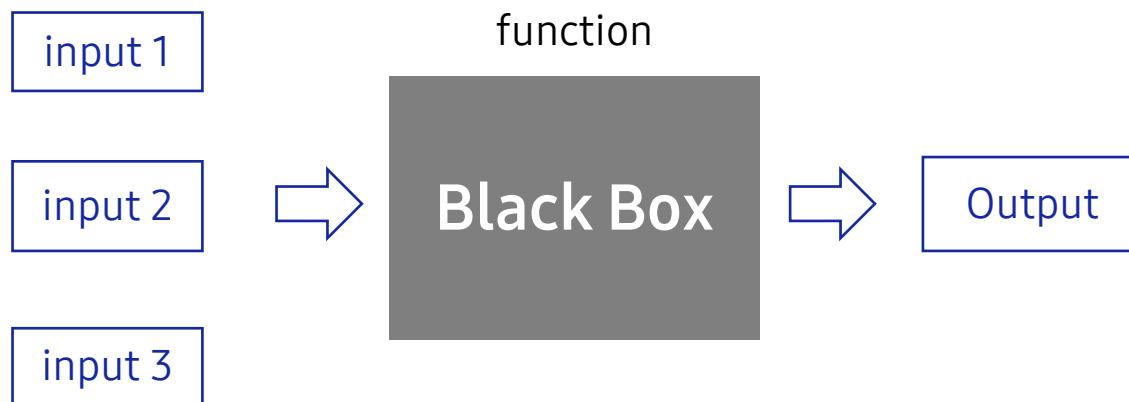
Write the entire code and the expected output results in the note.

| Let's code

1. ផ្សេងៗ Built-in (Built-in Functions)

1.1. ផ្សេងៗ Built-in ទៅសម្រាប់នូវការប្រើប្រាស់

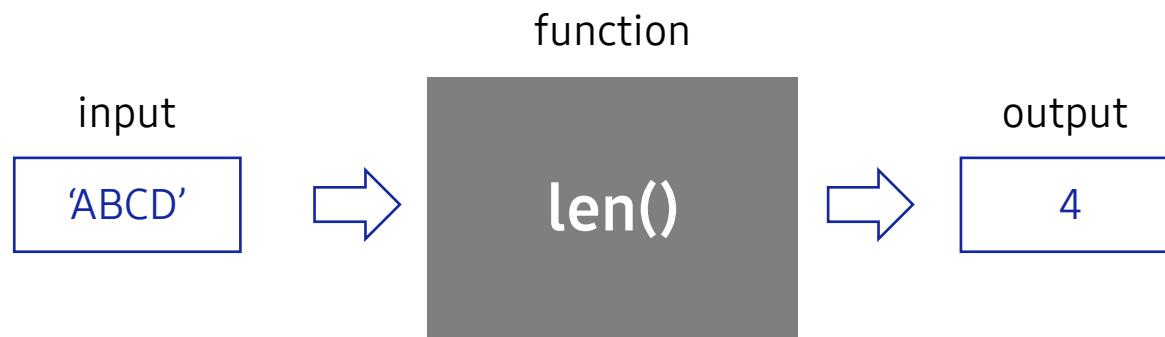
- | ដោយមានលក្ខណៈថា នូវការប្រើប្រាស់ជាផ្សេងៗ តាមទម្រង់ ដូចជា ការបែងចែក និងការសម្រាប់នូវការប្រើប្រាស់ជាបន្ទាល់។
- | នូវការប្រើប្រាស់ជាបន្ទាល់ គឺជាការបង្កើតរបស់ការប្រើប្រាស់ដែលមិនអាចពិនិត្យបាន។
- | ដើម្បីប្រើប្រាស់ជាបន្ទាល់ ត្រូវបានបង្កើតឡើងដោយប្រើប្រាស់ការបង្កើតនៃការប្រើប្រាស់ជាបន្ទាល់។



1. ផ្សេង Built-in (Built-in Functions)

1.1. ផ្សេង Built-in ទៅសម្រាប់ប្រើប្រាស់

- | ពីរយ៉ាង, ដើរតាក្នុងនៅក្នុងវា ផ្សេង len ត្រួតពិនិត្យរបស់ string ‘ABCD’, នឹងត្រូវសាមសិទ្ធិថាដើរីមិនអាចធ្វើបាន។
- | ផ្សេង print និង input បានប្រើប្រាស់ដោយខ្លួន។
- | ផ្សេងមែនស្ថាប័នីជាប្រព័ន្ធឌែលត្រូវបានត្រួតពិនិត្យនៅក្នុង Python ។
- | និង Python មិនមែនមានប្រព័ន្ធដែលត្រូវបានត្រួតពិនិត្យនៅក្នុង Python ។



1. ພັນຊັນ Built-in (Built-in Functions)

1.2. Python ແລະ built-in functions

Built-in functions in python				
abs	dict	help	min	setattr
all	dir	hex	next	slice
any	divmod	id	object	sorted
ascii	enumerate	input	oct	staticmethod
bin	eval	int	open	str
bool	exec	isinstance	ord	sum
bytearray	filter	issubclass	pow	super
bytes	float	iter	print	tuple
callable	format	len	property	type
chr	frozenset	list	range	vars
classmethod	getattr	locals	repr	zip
compile	globals	map	reversed	_import_
complex	hasattr	max	round	
delattr	hash	memoryview	set	

1. ພັງຊັນ Built-in (Built-in Functions)

1.3. ການນຳໃຊ້ built-in functions

| ໂດດຄໍາສິ່ງລຸ່ມນີ້ ແມ່ນການນຳໃຊ້ built-in functions.

```
1 abs(-100) # A function that returns an absolute value
```

```
100
```

```
1 min(200, 100, 300, 400) # A function that returns the minimum value among multiple elements
```

```
100
```

```
1 max(200, 100, 300, 400) # A function that returns the maximum value among multiple elements
```

```
400
```

- ▶ ພັງຊັນ abs ກັບຈຳນວນເຕັມ -100 ແລະ ສິ່ງຄໍາສືມບູນ 100 ອອກມາ.
- ▶ ພັງຊັນ min ຈະສິ່ງຄໍານີ້ອຍສຸດໃນບັນດາຈຳນວນເຕັມ 200, 100, 300 ແລະ 400.
- ▶ ພັງຊັນ max ຈະສິ່ງຄໍາໃຫຍ່ສຸດໃນບັນດາຈຳນວນເຕັມ 200, 100, 300 ແລະ 400..

1. ພັນຊັນ Built-in (Built-in Functions)

1.4. ການສົ່ງຄ່າກັບຂອງ built-in function

```
1 str1 = "FOO"          # Creates a string object in the format "FOO" or 'FOO'  
2 len(str1)            # Return the length of a string
```

3

```
1 type(str1)           # Return the type (type) of an object
```

str

```
1 id(str1)             # Return the ID value of an object
```

140168168956624

```
1 eval("100+200+300")    # Convert strings to numeric values and operators and evaluate them
```

600

- ▶ ພັນຊັນ id ຈະສົ່ງ ລະຫັດ (id) ຂອງຕົວປັນ str1 ທີ່ເກັບ string “FOO” .
- ▶ ພັນຊັນ type ຈະສົ່ງປະເພດຂໍ້ມູນຂອງບັນດາຕົວປັນທີ່ກ່ຽວຂ້ອງ.
- ▶ ພັນຊັນ len ຈະສົ່ງຄວາມຍາວຂອງ string ທີ່ເກັບໄວ້ໃນຕົວປັນ str1.
- ▶ ພັນຊັນ eval ຈະຮັບ string, ກຳນົດເນື້ອໃນຂອງ string, ປະເມີນ ແລະ ສົ່ງຄ່າທີ່ປະເມີນກັບ.
- ▶ ປະເມີນ string “100+200+300”, ມັນຈະຕິຄວາມເປັນ 100+200+300, ແລະ ສົ່ງຄ່າ 600 ອອກມາ.

1. ພັງຊັນ Built-in (Built-in Functions)

1.5. ພັງຊັນ Built-in ກັບການລຽງລໍາດັບ

| ໂຄດຄໍາສິ່ງລຸ່ມນີ້ແມ່ນການລຽງລໍາດັບຂອງ string ແລະ integer ແບບແຕ່ນ້ອຍຫາໃຫຍ່ ແລະ ແຕ່ໃຫຍ່ຫານ້ອຍ.

```
1 sorted("EABFD")      # Sort string
```

```
['A', 'B', 'D', 'E', 'F']
```

```
1 n_list = [200, 100, 300, 400, 50]
2 sorted(n_list)      # Sort the list in ascending order
```

```
[50, 100, 200, 300, 400]
```

```
1 sorted(n_list, reverse = True) # Sort the list in descending order
```

```
[400, 300, 200, 100, 50]
```

Line 1, 2

- ພັງຊັນ sorted ຈະຮັບຄໍາເປັນ string ແລະ ສິ່ງຄໍາເປັນ ຕົວອັກສອນທີ່ມີການລຽງລໍາດັບແຕ່ນ້ອຍຫາໃຫຍ່.
- ພັງຊັນ sorted ຍັງລຽງລໍາດັບ ແລະ ສິ່ງຄໍາທີ່ເປັນ list.
- reverse = true ແມ່ນເງື່ອນໄຂທີ່ເຮັດໃຫ້ລຽງລໍາດັບແຕ່ໃຫຍ່ຫານ້ອຍ.

1. ພັນຊັນ Built-in (Built-in Functions)

1.6. ພັນຊັນ id ແລະ ການຂຽນໂປຣແກຣມແບບວັດຖຸ

| ພັນຊັນ id ແລະ ການຂຽນໂປຣແກຣມແບບວັດຖຸ

- ▶ Python ແມ່ນພາສາໂປຣແກຣມແບບວັດຖຸ.
- ▶ ຄຸນສືມບັດຂອງວັດຖຸ ແລະ ພັນຊັນຕ່າງໆ ໃນການຂຽນໂປຣແກຣມ ຖືວ່າເປັນປັດໃຈຫຼັກຂອງພາສາວັດຖຸ.
- ▶ ວັດຖຸ Python ມີເອກະລັກສະເພາະຕົວທີ່ແຍກຄວາມແຕກຕ່າງຈາກວັດຖຸອື່ນໆ. ພັນຊັນ id ຈະສິ່ງລະຫັດປະຈຳຕົວທີ່ເປັນຈຳນວນຕັ້ນ. ເຊິ່ງຈະອະທິບາຍ ລະອຽດໃນ unit 21.

```
1 a_str = "Hello Python!"  
2 id(a_str)
```

2549478283440

```
1 n = 300  
2 id(n)
```

2549477426192

1. ພັງຊັນ Built-in (Built-in Functions)

1.7. ພັງຊັນ type

| ພັງຊັນ type ຈະສື່ງຄ່າທີ່ເປັນປະເພດຂໍ້ມູນຂອງວັດຖຸ. ມາຮຽນຮູ້ປະເພດຂໍ້ມູນຕ່າງໆໃນ Python.

```
1 type(123)
```

int

```
1 type('Hello String!')
```

str

```
1 type(120.3)
```

float

```
1 type([100, 300, 600])
```

list

1. ផ្សេងៗ Built-in (Built-in Functions)

1.8. ផ្សេងៗ eval

- | នីមួយៗបានដឹងពីរបៀបប្រើប្រាស់ការអនុវត្តន៍ eval ដើម្បីបង្កើតរូបរាងនៃការស្នើសុំការណែនាំ។
- | ផ្សេងៗ eval បានបង្កើតឡើងដើម្បីបានបញ្ជាក់ថាអ្នកបានបង្កើតការណែនាំដែលត្រូវបានបង្កើតឡើង។
- | ការប្រើប្រាស់ eval នឹងបង្កើតការណែនាំដែលត្រូវបានបង្កើតឡើង។

```
1 eval('10 + 20')      # The Python translator executes the sentence of 10 + 20
```

30

```
1 eval('(5 * 20) / 2') # The Python translator executes the sentence of (5 * 20) / 2
```

50.0

```
1 chr(65)            # The Unicode value 65 is the alphabet 'A', which the chr() function returns
```

'A'

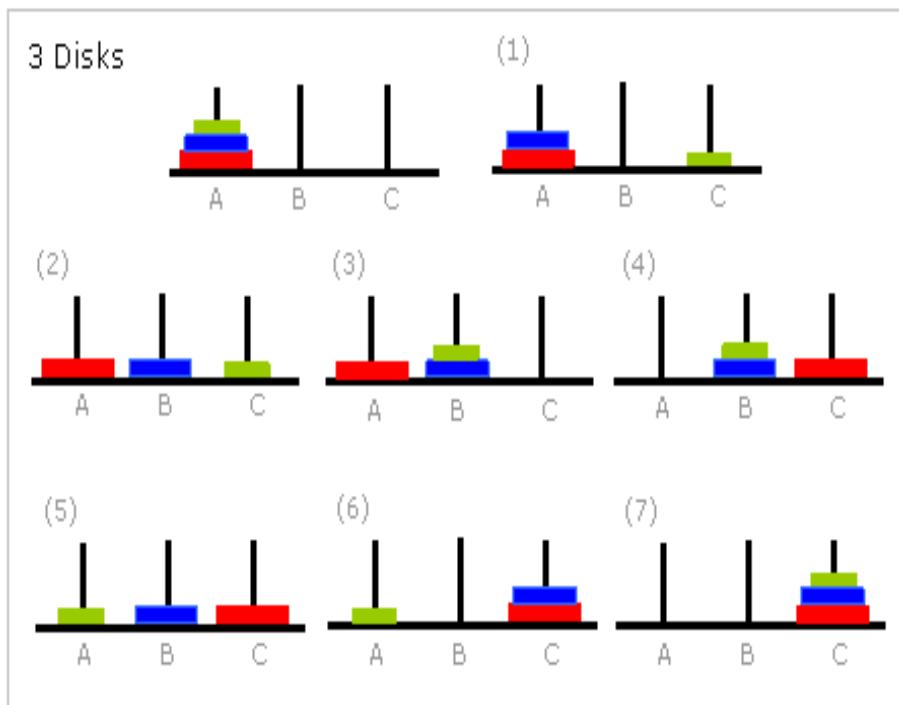
```
1 ord('A')           # Return the Unicode value 65 of the alphabet 'A'
```

65

2. បំណុលបញ្ហាគំណត់ Hanoi

2.1. ខ្សោយពាណិជ្ជកម្មរបស់លេខ

| Learn the step-by-step execution process of the Tower of Hanoi algorithm.



- ແກ້ໄຂបំណុលបញ្ហាគំណត់ Hanoi ដោយការងារបំផុត **recursive functions**.
- ឯជាលន្តា វំភ័យនឹងសាម disks ហើយសម្រេចនូវការងារចុះចុះ។ ស្ថាកាយ, disk ឲ្យយក (disk តិច) និង disk ឲ្យដាក់ (disk តិច) នឹងការងារចុះចុះ។
- រួចរាល់បំផុត, ឱ្យ disk ខាងក្រោម (disk តិច) និង disk ខាងលើ (disk តិច) នឹងការងារចុះចុះ។
- ឯជាលន្តា យក disk ក្នុងក្រុងបញ្ហាបាយទៅ B តាមការងារចុះចុះតិចពីក្រុង A និង រាយការងារចុះចុះតិចពីក្រុង C តាមការងារចុះចុះតិចពីក្រុង B ។
- ម៉ោង disk ខាងក្រោមនឹងការងារចុះចុះតិចពីក្រុង B, disk ខាងលើនឹងការងារចុះចុះតិចពីក្រុង C. [Figure 3]
- វំភ័យ, វិញ disk ខាងក្រោមនឹងការងារចុះចុះតិចពីក្រុង B, disk ខាងលើនឹងការងារចុះចុះតិចពីក្រុង C. [Figure 4]
- ឯជាលន្តា រាយការងារចុះចុះតិចពីក្រុង B និង រាយការងារចុះចុះតិចពីក្រុង C តាមការងារចុះចុះតិចពីក្រុង A. [Figure 5, 6, 7]

2. បំណុលខ្លួនការងារ Hanoi

2.1. ខ្លួនពាណិជ្ជកម្មរបស់អ្នក

| តូចមិនមែន disk 1 ដៃរៀង,

- ▶ 1. យើង disk ចាប់ផ្តើមពីតឱ្យ A ដោយចាប់ផ្តើមពីតឱ្យ C.

| តូចមិនមែន disk 2 ដៃរៀង,

- ▶ 1. យើង disk ចាប់ផ្តើមពីតឱ្យ B (តើវាមិនមែនទីតាំងទូទៅទេ).
- ▶ 2. យើង disk ចាប់ផ្តើមពីតឱ្យ C (តើវាមិនមែនទីតាំងទូទៅទេ).
- ▶ 3. យើង disk ចាប់ផ្តើមពីតឱ្យ C (តើវាមិនមែនទីតាំងទូទៅទេ).

| តូចមិនមែន disk 3 ដៃរៀង,

- ▶ 1. យើង disk 1, disk 2 ចាប់ផ្តើមពីតឱ្យ A ដោយចាប់ផ្តើមពីតឱ្យ B ដើម្បីធ្វើការចាប់ផ្តើមមែន 2 disk,
- ▶ 2. យើង disk 3 ដោយចាប់ផ្តើមពីតឱ្យ C ដើម្បីធ្វើការចាប់ផ្តើមមែន 1 disk,
- ▶ 3. យើង disk 1, disk 2 ចាប់ផ្តើមពីតឱ្យ B ដោយចាប់ផ្តើមពីតឱ្យ C ដើម្បីធ្វើការចាប់ផ្តើមមែន 2 disk.

2. ບັນຫາຂອງຫຳຄອຍ Hanoi

2.2. ຂັ້ນຕອນວິທີຂອງຫຳຄອຍ Hanoi

| ຖ້າມີ n disk,

- ▶ 1. ຍ້າຍ n-1 disk ໄປ B (ຄືກັບມີ disk n-1 ແຜ່ນ).
- ▶ 2. ຍ້າຍ disk ສຸດທ້າຍ ໄປ C (ຄືກັບມີ disk 1 ແຜ່ນ).
- ▶ 3. ຍ້າຍ n-1 disk ຈາກ B ໄປ C (ຄືກັບມີ disk n-1 ແຜ່ນ).

| ຖ້າພວກເຮົາໃຊ້ໂຄງສ້າງນີ້ໂດຍນຳໃຊ້ ພົງຊັ້ນເອັນຊ້າ (recursive function) ຈະເປັນດັ່ງລຸ່ມນີ້:

```
1 hanoi(n-1, from_, via_, to_)
2 print("Number {}, {} -> {}".format(n, from_, to_))
3 hanoi(n-1, via_, to_, from_)
```

| Pair programing



Pair Programming Practice

| ແນວທາງ, ກິນໄກ ແລະ ແຜນສຸກເສີນ

ການຈັບຄຸ້ຂຽນໂປຣແກຣມ ເປັນການຈັບຄຸ້ຂອງນັກຮຽນເພື່ອຮັດວຽກມອບໝາຍ, ນັກຮຽນຄວນມີແຜນ ແລະ ສາມາດປ່ຽນແທນກັນໄດ້ ໃນ ກໍາລະນີມີຜູ້ໃຫ້ນີ້ບໍ່ສາມາດເຂົ້າຮ່ວມຮັດວຽກມອບໝາຍໄດ້ບໍ່ວ່າໃນກໍາລະນີໃດກຳຕາມ ເຊິ່ງບັນຫາເລື່ອນັ້ນຕ້ອງຮັດໃຫ້ຈະເຈັ້ງ ແລະ ກຳບໍ່ແມ່ນ ຄວາມຝຶດຂອງນັກຮຽນທີ່ຈັບຄຸ້ບໍ່ດີ.

| ຈັບຄຸ້ທີ່ຄ້າຍຄືກັນ, ບໍ່ຈໍາເປັນເຫົ້າທຽມກັນ, ຄວາມສາມາດເປັນຄຸ້ຮ່ວມງານ

ການຈັບຄຸ້ຂຽນໂປຣແກຣມ ຈະໄດ້ຮັບຜົນທີ່ກຳຕໍ່ເນື່ອນັກຮຽນມີຄວາມສາມາດຄ້າຍຄືກັນ ຫາຍວ່າ ບໍ່ຈໍາເປັນຕ້ອງມີຄວາມສາມາດຄືກັນກຳໄດ້, ແຕ່ວ່າ ການຈັບຄຸ້ ນັກຮຽນທີ່ມີຄວາມສາມາດແຕກຕ່າງກັນຫຼາຍ ກໍຈະຮັດໃຫ້ບໍ່ສົມດຸນກັນ. ຄຸສອນຮຸດດີວ່າ ການຈັບຄຸ້ກັນບໍ່ແມ່ນ ຍຸດທະສາດ “ແບ່ງເພື່ອເອົາຊະນະ” ແຕ່ເປັນຄວາມ ພະຍາຍາມຮັດວຽກຮ່ວມກັນຂອງນັກຮຽນໃຫ້ປະສິບຜົນສໍາເລັດ. ຄຄວນທີ່ກາເວັ້ນການຈັບຄຸ້ກັນລະຫວ່າງນັກຮຽນອ່ອນ ແລະ ນັກຮຽນເກົ່າງ.

| ກະຕຸ້ນນັກຮຽນໂດຍການໃຫ້ສິ່ງຈຸງໃຈພື້ນເສດ

ຂໍສະເໜີແຮງຈຸງໃຈທີ່ຮັດໃຫ້ນັກຮຽນຈັບຄຸ້, ໂດຍສະເພາະນັກຮຽນທີ່ມີຄວາມສາມາດສຸງ. ບາງຄຸສອນໄດ້ພື້ນວ່າ ການຈັບຄຸ້ຮັດວຽກມອບໝາຍ ແມ່ນມີ ປະໂຫຍດ ສໍາລັບໜຶ່ງ ຫຼື ສອງວຽກມອບເທົ່ານັ້ນ



Pair Programming Practice

| ចំណាំការងារប់ព័ីននៃការសម្រេច

សៀវភៅទាយសាំລាបតុអំពើដែលបានបង្ហាញការងារប់ព័ីននៃការសម្រេច តុនូវបែងចែករបស់ខ្លួន និងបង្ហាញការងារប់ព័ីននៃការសម្រេច ដើម្បីស្វែងរកពិតាទីតាំងនៃការងារប់ព័ីននៃការសម្រេច។ ដូច្នេះ ការងារប់ព័ីននៃការសម្រេច គឺជាប្រព័ន្ធដែលបានបង្ហាញការងារប់ព័ីននៃការសម្រេច និងបង្ហាញការងារប់ព័ីននៃការសម្រេច ដើម្បីស្វែងរកពិតាទីតាំងនៃការងារប់ព័ីននៃការសម្រេច។

| សម្រាប់រាយការណាមីនិយោគ

សម្រាប់រាយការណាមីនិយោគ គឺជាប្រព័ន្ធដែលបានបង្ហាញការងារប់ព័ីននៃការសម្រេច និងបង្ហាញការងារប់ព័ីននៃការសម្រេច ដើម្បីស្វែងរកពិតាទីតាំងនៃការងារប់ព័ីននៃការសម្រេច។ ដូច្នេះ ការងារប់ព័ីននៃការសម្រេច គឺជាប្រព័ន្ធដែលបានបង្ហាញការងារប់ព័ីននៃការសម្រេច និងបង្ហាញការងារប់ព័ីននៃការសម្រេច ដើម្បីស្វែងរកពិតាទីតាំងនៃការងារប់ព័ីននៃការសម្រេច។

Q1.

จำนวนที่มีค่าเดียว e ซึ่งเรียกว่า Euler จำนวน Euler ค่าถูกกำหนดโดย Napier และจำนวน irrational ที่ถูกกำหนดให้เป็นพื้นฐานของ logarithm. ดังนั้นในสูตรลุ่มนี้.

$$(e = 1 + 1/1! + 1/2! + 1/3! + \dots + 1/n!)$$

ในสูตรข้างต้นนี้ กำหนด k ให้ฟังก์ชันที่ชื่อ factorial(k). นอกจากนี้ กำหนดฟังก์ชัน euler(n) เพื่อส่งคืนผลบวกของ $1/0! + 1/1! + \dots + 1/n!$ ถ้า $euler(20)$ ที่มีผลลัพธ์ตามที่แสดง และ ส่งคืนรับอุปกรณ์ที่ดังกล่าว. (ต้องใช้ recursive function.)

Output example

```
eular(20) = 2.71828
```