Week 11 Tutorial

COMP10001 – Foundations of Computing

Semester 2, 2025

Clement Chau

- Recursion

Agenda

1. Week 11 Discussion – **Tutorial sheet** (~ 70 mins)
2. One-on-one Q&A (~ 40 mins)

11 (13/10)	Ethics & HTML	Ethics & HTML (continued)	Advanced lecture	Week 11 tutorial sheet Week 11 tutorial solutions	• Project 2 due (17/10 at 6pm)

Project 2 (15%) due (17/Oct, Friday at 6 pm)

Ed worksheets 16-17 due (20/Oct, Monday at 6 pm)

Revision: Recursion

sem1-2025 > week-11 > find_max_num_recursion.py > ...

```python
def find_max(numbers):
    """
    Find the maximum number in a list using recursion.
    """

    # Base case: if the list has only one element, return that element.
    if len(numbers) == 1:
        return numbers[0]

    # Recursive case: compare the first element with the maximum of the rest of the list and return the maximum.
    return max(numbers[0], find_max(numbers[1:]))

find_max([1, 2, 3, 4, 5])  # Output: 5.
```

base case

recursive case

TuteSheet W11 – Recursion

***Let's think about** Factorial*

*5! = 5 * 4 * 3 * 2 * 1*

*4! = 4 * 3 * 2 * 1*

3! …

```python
def factorial(n):
    if n == 0:                    # Base case
        return 1
    return n * factorial(n - 1) # Recursive case
```

- **base case:** where the function has reached the smallest input or **simplest version** of the problem; **stops recursing** and returns an answer
- **recursive case:** where the function calls **itself** with a reduced or simpler input

How it works for factorial(5):

*factorial(5) → 5 * factorial(4)*

*factorial(4) → 4 * factorial(3)*

*factorial(3) → 3 * factorial(2)*

*factorial(2) → 2 * factorial(1)*

*factorial(1) → 1 * factorial(0)*

factorial(0) → 1 (base case)

Then it returns:

1

1 * 1 = 1

2 * 1 = 2

3 * 2 = 6

4 * 6 = 24

5 * 24 = *120*

Ed Worksheet 17 > Intro to Recursion > Factorial

https://www.geeksforgeeks.org/program-for-factorial-of-a-number/

TuteSheet W11 – Recursion

3!

```
def factorial(3):
    if 3 == 0:
        return 1
    return 3 * factorial(3 - 1)
```

*returns 3 * factorial (2) =* ***6***

2

```
def factorial(2):
    if 2 == 0:
        return 1
    return 2 * factorial(2 - 1)
```

*returns 2 * factorial(1) =* ***2***

1

```
def factorial(1):
    if 1 == 0:
        return 1
    return 1 * factorial(1 - 1)
```

*returns 1 * factorial(0) =* ***1***

0

```
def factorial(0):
    if 0 == 0:
        return 1
    return 0 * factorial(0 - 1)
```

returns ***1***

TuteSheet W11 – Exercises Q1

1. Recap the basics of recursion:

   - *What makes a function recursive?* Recursion is where a function calls itself [redacted] to solve a problem. Rather than using a loop to iterate through a sequence or repeat an action, a recursive function usually [redacted] with a smaller or broken-down version of the input [redacted] it reaches the answer.
   - *What are the two parts of a recursive function?* Recursive functions include a "recursive case", where the function calls itself with a [redacted] or [redacted] input; and a "base case" where the function has reached the smallest input or simplest version of the problem: it [redacted] recursing and [redacted] an answer.
   - *In what cases is recursion useful?* Recursion is useful where an iterative solution would require nesting of loops proportionate to the size of the input, such as the powerset problem. Some algorithms you will learn about in future subjects depend on recursion, and it can be a powerful technique when trying to sort data.
   - *When should we be cautious about using recursion?* There will often be an equally elegant iterative solution, and since function calls are expensive, it's often more efficient to use the iterative approach.

TuteSheet W11 – Exercises Q1(a)

Now, study the following mysterious functions. For each one, answer the following questions:

- Which part is the base case?
- Which part is the recursive case?
- What does the function do?

(a)
```
def mystery(x):
    if len(x) == 1:
        return x[0]
    else:
        y = mystery(x[1:])
        if x[0] > y:
            return x[0]
        else:
            return y
```

`if` block : base case

`else` block : recursive case

The function returns ***the largest element in the list/tuple.*** *If the input is an empty list, the function never reaches the base case so a `RecursionError` is raised.*

TuteSheet W11 – Exercises Q1(b)

Now, study the following mysterious functions. For each one, answer the following questions:

- Which part is the base case?
- Which part is the recursive case?
- What does the function do?

(b)

```
def mistero(x):
    a = len(x)
    if a == 1:
        return x[0]
    else:
        y = mistero(x[:a//2])
        z = mistero(x[a//2:])
        if z > y:
            return z
        else:
            return y
```

`if` block : base case

`else` block : recursive case

*Like (a), this function returns **the largest element in the list/tuple**. This function uses two recursive calls, while the first uses one. There's no difference in the calculated output.*

TuteSheet W11 – Exercises Q2

2. Here is a classic example of a recursive function, finding the nth Fibonacci number, and a recursion visualisation tree for this function:

Fibonacci number (sequence) *: each number is equal to the sum of the preceding two numbers*

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377

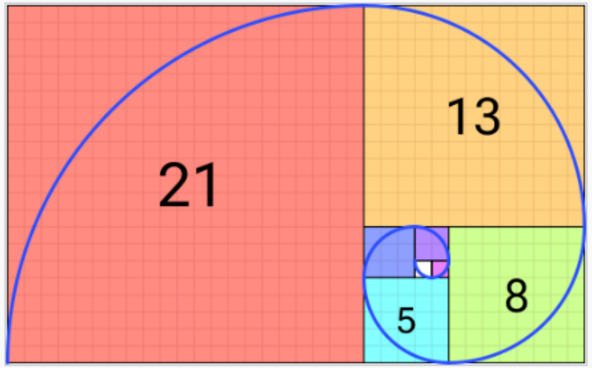
$$F_0 = 0, \quad F_1 = 1,$$

and

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

for $n > 1$.

```python
def fib(n):
  if n in [0, 1]:
    return n
  return fib(n - 1) + fib(n - 2)
```



The Fibonacci spiral: an approximation of the golden spiral created by drawing circular arcs connecting the opposite corners of squares in the Fibonacci tiling (see preceding image)

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

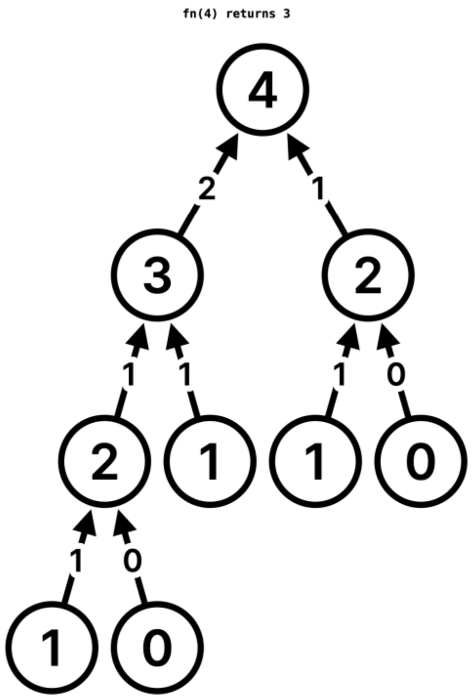
TuteSheet W11 – Exercises Q2

2. Here is a classic example of a recursive function, finding the nth Fibonacci number, and a recursion visualisation tree for this function:

Fibonacci number (sequence) *: each number is equal to the sum of the preceding two numbers*

F_0	F_1	F_2	F_3	F_4	F_5
0	1	1	2	3	5

```python
def fib(n):
  if n in [0, 1]:
    return n
  return fib(n - 1) + fib(n - 2)
```



recursion tree

- *Each node : a function call*
- *Label: input*
- *Arcs: returned value*

https://recursion.vercel.app/

TuteSheet W11 – Exercises Q2

recursion tree

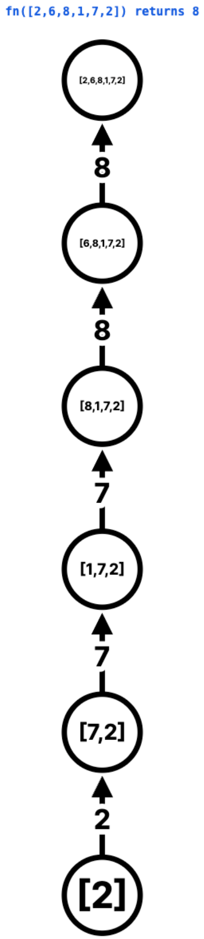
- *Each node : a function call*
- *Label: input*
- *Arcs: returned value*

Draw (by hand) a recursion visualisation tree of the inputs and outputs of `mystery(x)` and `mistero(x)`, from the previous question, for the input `x = [2, 6, 8, 1, 7, 2]`.

(a)
```python
def mystery(x):
    if len(x) == 1:
        return x[0]
    else:
        y = mystery(x[1:])
        if x[0] > y:
            return x[0]
        else:
            return y
```

returns ***the largest element in the list/tuple.***

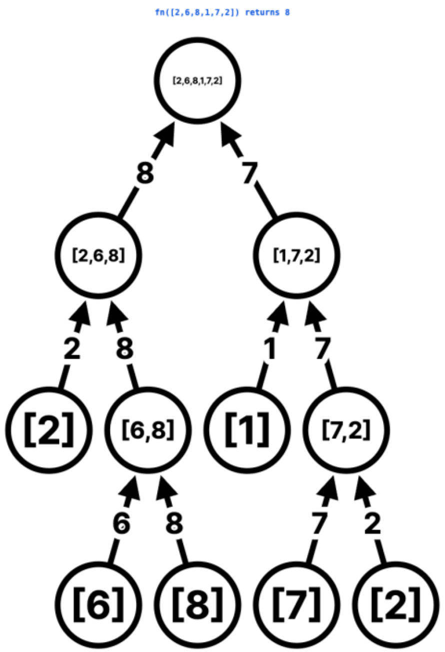
- ***base*** *: only 1 element, return it*
- ***recursive****: first element vs. the max of the rest, return the larger one*



(b)
```python
def mistero(x):
    a = len(x)
    if a == 1:
        return x[0]
    else:
        y = mistero(x[:a//2])
        z = mistero(x[a//2:])
        if z > y:
            return z
        else:
            return y
```

using ***divide and conquer*** ***(splits the list in half each time)***

- ***recursive****: Divide list into two halves*



TuteSheet W11 – Exercises Q3

3. This version of the change-making problem asks if it is possible to make some `amount` from a given selection of coin values, possibly using multiple coins of a particular value.

   For example, if my amount is `23` and I have coins with values `[3, 6, 8]`, then I can make `23` with one `3`, two `6` and one `8`. But if my amount is `11` and I have coins with values `[2, 8, 6]`, then I am unable to make `11`.

   11 < (2 + 8 + 6)

   Fill in the blanks of the `can_make_change(amount, coins)` function.

```
def can_make_change(amount, coins):
    # base case: success
    if amount == 0:
        return True

    # base case: failure
    if amount < 0 or len(coins) == 0:
        return False

    # recursive case: handle two possibilities, either:
    # 1. another of this coin value gets used, or
    # 2. we don't need another coin of this value
    coin = coins[-1]
    return (can_make_change(amount - coin, coins)
            or can_make_change(amount, coins[:-1]))
```

23 = (3 + 6 + 6 + 8)

23 – (3 + 6 + 6 + 8) = 0

23 – 8 = 15, ***[3, 6]***

15 - 6 = 9, [3, 6]

9 – 6 = 3, [3, 6]

3 – 6 = -3 (false)

3 – 3 = 0 , [3] (true)

TuteSheet W11 – Exercises Q3

3. This version of the change-making problem asks if it is possible to make some `amount` from a given selection of coin values, possibly using multiple coins of a particular value.

   For example, if my amount is `23` and I have coins with values `[3, 6, 8]`, then I can make `23` with one `3`, two `6` and one `8`. But if my amount is `11` and I have coins with values `[2, 8, 6]`, then I am unable to make `11`.

   Fill in the blanks of the `can_make_change(amount, coins)` function.

Pre-defined templates

Custom

Global variables

=

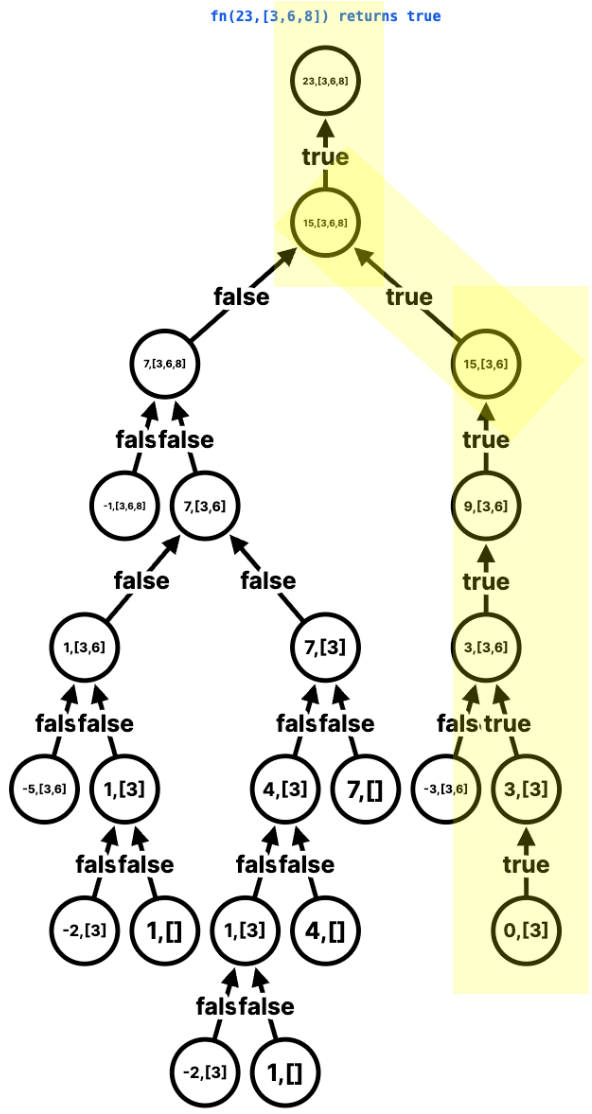
=

Recursive function python

```python
def fn(amount, coins):
    if amount == 0:
        return True
    if amount < 0 or len(coins) == 0:
        return False
    coin = coins [-1]
    return (fn(amount-coin, coins) or fn(amount,
coins[:-1]))
```

```
fn(23, [3, 6, 8])
```

Run



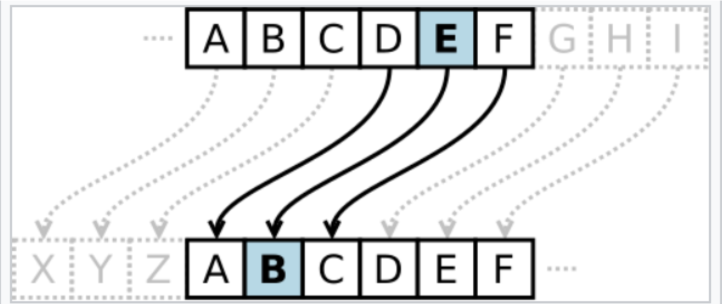
TuteSheet W11 – Practice Programming Q1

Challenge: You've found a secret message:

```
secret_message.txt
```

```
erkbvl ur kbvd tlmexr:
gxoxk zhggt zbox rhn ni
gxoxk zhggt exm rhn whpg
gxoxk zhggt kng tkhngw tgw wxlxkm rhn
gxoxk zhggt ftdx rhn vkr
gxoxk zhggt ltr zhhwurx
gxoxk zhggt mxee t ebx tgw ankm rhn
```

All that you know about the message is that it is encrypted by a basic shift cipher (also known as a Caesar cipher, where each letter is shifted by some constant number of places in the alphabet), any alphabetic character in the message is lowercase, and that it contains the string segment `'desert'`.



the cipher is the plain alphabet rotated left or right by some number of positions.
*The cipher illustrated here uses **a left shift of 3 ,** equivalent to **a right shift of 23***

Caesar cipher: https://en.wikipedia.org/wiki/Caesar_cipher
https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/

TuteSheet W11 – Practice Programming Q1

All that you know about the message is that it is encrypted by a basic shift cipher (also known as a Caesar cipher, where each letter is shifted by some constant number of places in the alphabet), any alphabetic character in the message is lowercase, and that it contains the string segment `'desert'`.

encryption

a left shift of 3

shift = -3

Plain	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

decryption

a right shift of 3

shift = 3

<u>*In this example,*</u>

shift = ??

Plain	a	b	c	d	e	f	…	r	s	t	…	y	z
Cipher													

Hint : "desert"

Don't know the "shift"

Which version contains the "desert"?

TuteSheet W11 – Practice Programming Q1

Write a function to decrypt the message that takes an `infilename`, `outfilename` and `segment` (all strings, and you can assume that all files exist). You can use a brute-force approach (try all possible values) to guess the number to shift by. You might find the functions `ord(character)` and `chr(number)` useful!

- *Hint : "desert"*
- *Don't know the "**shift**"*
- ***we try all 26 alphabet possibilities** and*
- *check which version contains the "desert"?*

```
ord("a")              97
ord("d")              100
chr(ord("a"))         'a'
chr(ord("a")+3)       'd'
```

```
shift = -3
letter_number = (ord("d") + shift - ord("a")) % 26
chr(ord("a") + letter_number)
```

'a'

https://docs.python.org/3/library/functions.html#ord
https://docs.python.org/3/library/functions.html#chr

TuteSheet W11 – Practice Programming Q1

Write a function to decrypt the message that takes an `infilename`, `outfilename` and `segment` (all strings, and you can assume that all files exist). You can use a brute-force approach (try all possible values) to guess the number to shift by. You might find the functions `ord(character)` and `chr(number)` useful!

	Plain	a	b	c	d	e	f	…	r	s	t	…	y	z
input	Cipher				*w*	*x*			*k*	*l*	*m*			

output

decryption

a right shift of 7

```python
def caesar_cipher_alphabet(shift):
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    alphabet_size = len(alphabet) #26
    cipher = ""

    for letter in alphabet:
        letter_number = (ord(letter) + shift - ord("a")) % alphabet_size
        cipher += chr(ord("a") + letter_number)
```

```python
caesar_cipher_alphabet(-7)
```

```
Plain : abcdefghijklmnopqrstuvwxyz
Shift : -7
Cipher: tuvwxyzabcdefghijklmnopqrs
```

shift = -7

shift = 7

TuteSheet W11 – Practice Programming Q1

Write a function to decrypt the message that takes an `infilename`, `outfilename` and `segment` (all strings, and you can assume that all files exist). You can use a brute-force approach (try all possible values) to guess the number to shift by. You might find the functions `ord(character)` and `chr(number)` useful!

```
ALPHABET_SIZE = 26

def shift_cipher_decoder(infilename, outfilename, segment):
```

```
erkbvl ur kbvd tlmexr:
gxoxk zhggt zbox rhn ni
gxoxk zhggt exm rhn whpg
gxoxk zhggt kng tkhngw tgw wxlxkm rhn
gxoxk zhggt ftdx rhn vkr
gxoxk zhggt ltr zhhwurx
gxoxk zhggt mxee t ebx tgw ankm rhn
```

```
lyrics
never
never
never                              rt you
never
never
never                             you
```


```
shift_cipher_decoder("encrypted_message.txt", "decrypted_message.txt", "desert")
```

Input file: Ciper ***Output file: Plain*** ***Hint***

In Week 12

Please bring your laptop if you can.

- **Ethics, Algorithms, HTML**
- **Programming with GitHub Copilot in VS code**
  - ✓ Install and setup
  - ✓ Prompt Engineering
  - ✓ GitHub page

Independent Work

- **Next due dates:**
  - **Project 2** is due **this Friday, October 17th, 6pm**.
    - Project 2 is (considerably) more difficult than Project 1. **Start early**.
  - Final Ed Worksheets **16** and **17** is **due next Monday, October 20th, 6pm**.
- **Raise your hand** if you have any questions!

Scan here for annotated slides