Week 11 Tutorial

COMP10001 – Foundations of Computing

Semester 1, 2025

Clement Chau

- Recursion
- Binary, Octal, Hexadecimal *(Computational Counting)*

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

Exercise 1, identify and compare

B = Base case

R = Recursive case

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
```
B

R

(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
```
B

R

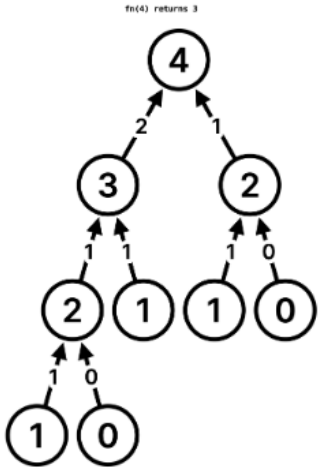
A: *The `if` block is the base case, and the `else` block is the recursive case. The function returns the largest element in the list/tuple. If the input is an empty list, the function never reaches the base case so a `RecursionError` is raised.*

A: *The `if` block is the base case, and the `else` block is the 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.*

Exercise 2, draw a recursion tree

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

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



Recursion visualisation of `fib(4)` from `https://recursion.vercel.app/`.

Each node in the recursion tree represents a function call and is labelled with the input, the arcs show the 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]`.

Exercise 3, is there enough change?

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.

```
def can_make_change(amount, coins):
    # base case: success
    [ 1 ]
        return True

    # base case: failure
    if amount < 0 or len(coins) == 0:
        [ 2 ]

    # 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( [ 3 ] , coins)
            or can_make_change(amount, coins[:-1]))
```

Answer:

A: *Blank 1:* `if amount == 0:`

Blank 2: `return False`

Blank 3: `amount - coin`

Revision: Binary and Octal

- Binary
  - Consists of 0's and 1's
  - Form is 2^x, where x is the **position (from 0) from the right**.
  - e.g. **1011 = 1 x 2^3 + 0 x 2^2 + 1 x 2^1 + 1 x 2^0 = 8 + 0 + 2 + 1 = 11**
- Octal
  - Similar to the decimal system, but with **base 8**.
  - e.g. For the decimal **74**,
    - Decimal: **74_{10} = 70 + 4 = 7 x 10^1 + 4 x 10^0**
    - Octal: **74_8 = 64 + 8 + 2 = 1 x 8^2 + 1 x 8^1 + 2 x 8^0 => 112**
  - e.g. For the binary **11111001**,
    - Make sure the binary is in **multiples of 3**, otherwise modify it => **011111001**
    - **Divide** the binary into **lengths of 3**, then **evaluate the decimal**:
      - **011** in decimal is **3**
      - **111** in decimal is **7**
      - **001** in decimal is **1**
      - Therefore, final answer is **371_8**

Revision: Hexadecimal

- Similar to the decimal and octal system, but with **base 16**.
- e.g. For the decimal **78**,
  - In binary this is $2^6 + 2^3 + 2^2 + 2^1$ => 01001110
  - Splitting into **lengths of 4**:
    - **0100** -> $\mathbf{4}_{10}$ -> $\mathbf{4}_{16}$
    - **1110** -> $\mathbf{14}_{10}$ -> $\mathbf{E}_{16}$
    - So, $\mathbf{78}_{10}$ == $\mathbf{4E}_{16}$
- Order of converting:
  - **Octal** <-> **Binary** <-> **Hexadecimal**
  - **Octal** <-> **Decimal** <-> **Binary** <-> **Hexadecimal**

Decimal to Hexadecimal Table

MATH MONKS

Decimal (Base 10)	Hexadecimal (Base 16)	Binary (Base 2)
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Exercise 4(a), 4(b): Computational Counting

Dec	Bin	Oct	Hex	Dec	Bin	Oct	Hex
0				11			
1				12			
2				13			
3				14			
4				15			
5				16			
6				17			
7				18			
8				19			
9				20			
10				21			

Table 1: Count from 1 to 21 using different number systems.

Power	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal								
Binary								

Answer to Exercise 4(a), 4(b):

Dec	Bin	Oct	Hex	Dec	Bin	Oct	Hex
0	0	0	0	11	1011	13	B
1	1	1	1	12	1100	14	C
2	10	2	2	13	1101	15	D
3	11	3	3	14	1110	16	E
4	100	4	4	15	1111	17	F
5	101	5	5	16	10000	20	10
6	110	6	6	17	10001	21	11
7	111	7	7	18	10010	22	12
8	1000	10	8	19	10011	23	13
9	1001	11	9	20	10100	24	14
10	1010	12	A	21	10101	25	15

Table 1: Count from 1 to 21 using different number systems.

Power	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal	128	64	32	16	8	4	2	1
Binary	10000000	1000000	100000	10000	1000	100	10	1

Exercise 4(c), deciphering hex colour codes

(c) Converting a base N number to decimal is not hard: it is simply the sum of $decimalDigit * base^{position}$ for each digit. For example, $A5B$ in hexadecimal (i.e. $A5B_{16}$) is equivalent to 2651 in decimal (i.e. 2651_{10}). This is because $10 * 16^2 + 5 * 16^1 + 11 * 16^0 = 2651$.

One common use of hexadecimal numbers is in hex colour codes, where colours are represented as a combination of red, green, and blue values, each ranging from 00 to FF (or 0 to 255 in decimal). Convert the following hex codes into their decimal red, green, and blue components and then guess the colour. The first one is done for you:

Hex code	**Red**	**Green**	**Blue**	**Guess Colour**
#FFFF00	255	255	0	Yellow
#000000	0			
#008080	0			
#BD00FF	189			

Answer to Exercise 4(c):

Hex code	Red	Green	Blue	Guess Colour
#FFFF00	255	255	0	Yellow
#000000	0	0	0	Black
#008080	0	128	128	Teal
#BD00FF	189	0	255	Pink-ish Purple

A: *For converting #FFFF00 into decimal RBG values we can see the first two digits are FF so we can do the calculation to convert this to decimal. Since hexadecimal F represents 15 in decimal, we multiply 15 by the base 16 to the power at that position:* $15 * 16^1 + 15 * 16^0 = 15 * 16 + 15 * 1 = 240 + 15 = 255$. *This means that the decimal value for the red is 255. We can do the same thing for the green value. And for the blue,* $0 * 16^1 + 0 * 16^0 = 0$ *so we have 0 for the blue value. In additive light colour theory, red and green make yellow!*
For converting #000000 into decimal RBG values we can see the first two digits are 00 so red is 0. The next two are also 00 so green is 0 and same with blue being 0. This results in the colour Black!
For converting #008080 into decimal RBG values we can see the first two digits are 00 so red is 0. The next two digits are 80 so we calculate $8 * 16^1 + 0 * 16^0$ *and get* 128 *so green is 128. The next two digits are also 80 so there is 128 for blue as well. Thinking about light additive colour theory, moderate amounts of green and blue will result in a teal colour.*
For converting #BD00FF into decimal RBG values, we have the first two digits being BD. B represents 11 in decimal and D represents 13. Then we calculate $11*16^1+13*16^0 = 11*16+13*1 = 176+13 = 189$ *so the red value is 189. The next two digits are 00 so the green value is 0 and finally, the FF is* $15*16^1+15*16^0 = 15*16+15*1 = 240+15 = 255$ *so the blue value is 255. Thinking about light additive colour theory, we have a fair amount of red and a lot of blue so we would expect this to be some kind of purple or pink.*

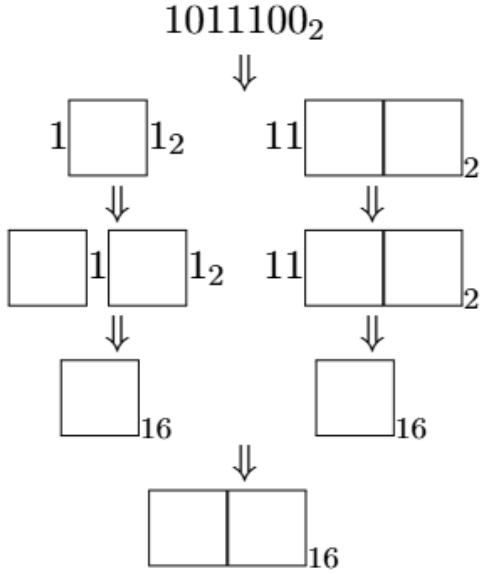
How to convert between bases in Python

- Decimal <-> Binary
  - **bin()** e.g. **bin(7)** returns **"0b111"** (equivalent to**"0b0111"**)
  - **int(num, base)** e.g. **int("1011", 2)** returns **7**
  - **int(0b0111)** returns **7**
- Decimal <-> Octal
  - **oct()** e.g. **oct(63)** returns **"0o77"**
  - **int(num, base)** e.g. **int("77", 8)** returns **63**
  - **int(0o77)** returns **63**
- Decimal <-> Hexadecimal
  - **hex()** e.g. **hex(230)** returns **"0xe6"** (equivalent to **"0xE6"**)
  - **int(num, base)** e.g. int**("E6", 16)** returns **230**
  - **int(0xE6)** returns **230**

Exercise 5, convert binary to hexadecimal

5. Convert the following binary numbers into hexadecimal. If you're stuck, take a look at exercise 2a and think about this question: how many binary digits are required to represent a hexadecimal digit?

   (a) Convert the binary number 1011100 to hexadecimal by filling in the boxes in the following diagram with a single digit:



   (b) Convert the binary number 111101001 into hexadecimal using a method like the one shown above.

Answers to Exercise 5:

5(a)

$$1011100_2$$
$$\Downarrow$$
$$101_2 \quad 1100_2$$
$$\Downarrow \quad \Downarrow$$
$$0101_2 \quad 1100_2$$
$$\Downarrow \quad \Downarrow$$
$$5_{16} \quad C_{16}$$
$$\Downarrow$$
$$5C_{16}$$

5(b)

$$111101001_2$$
$$\Downarrow$$
$$1_2 \quad 1110_2 \quad 1001_2$$
$$\Downarrow \quad \Downarrow \quad \Downarrow$$
$$0001_2 \quad 1110_2 \quad 1001_2$$
$$\Downarrow \quad \Downarrow \quad \Downarrow$$
$$1_{16} \quad E_{16} \quad 9_{16}$$
$$\Downarrow$$
$$1E9_{16}$$

Step 1: separate into 4-bit sequences
Step 2: Add leading zeroes to make them all four bits long
Step 3: Directly convert binary numbers (0000-1111) into hexadecimal numbers (0-F)
Step 4: Combine hexadecimal numbers together, retaining place value of the original binary sequence

Programming Practice

Problem 1, recursion!

1. Write a recursive function to flatten a nested list of integers. For example,

```
>>> flatten_list([[0, 1], 2, [3, 4, [5, 6, [7], 8]]])
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

Answer:

A:

```
def flatten_list(input_list):
    """Takes a nested list and returns a flattened version of it."""
    # base case: if the data is an integer, return it in a list
    if type(input_list) == int:
        return [input_list]

    # recursive case: if the data is a list
    # then flatten each element and combine
    result = []
    for item in input_list:
        result.extend(flatten_list(item))
    return result
```

Problem 2, convert hexadecimal into RGB

2. Write a Python function to convert a hexadecimal colour code into its corresponding RGB integer values. The function should accept a hex code string (e.g. `'#1E90FF'`) and return a tuple containing the red, green, and blue values as integers.

```
>>> hex_to_rgb('#1E90FF')
(30, 144, 255)
```

A:

Answer:

```
HEX_BASE = 16

def hex_to_rgb(hex_code):
    # Remove the '#' from the start
    hex_code = hex_code.lstrip('#')

    # Convert the hex code to RGB values
    colour = [int(hex_code[i:i+2], HEX_BASE)
              for i in range(0, len(hex_code), 2)]

    # Return the RGB values as a tuple
    return tuple(colour)
```

Independent Work

- **Next due dates:**
  - **Project 2** is due **this Friday, May 23rd, 6pm**.
    - Project 2 is (considerably) more difficult than Project 1. **Start early**.
  - (Final) Ed Worksheets **16** and **17** is **due next Monday, May 26th, 6pm**.
- **Project 1 marks should have been out by now.**
  - If you have any questions regarding your marks for Project 1, my email is **clement.chau@unimelb.edu.au**.
- **Raise your hand** if you have any questions!

Scan here for annotated slides