INF1002
Programming Fundamentals
Lecture 5: Recursion and Docstring

Zhang Zhengchen
zhengchen.zhang@singaporetech.edu.sg

Review

- File I/O
  - read, readline, readlines
  - write, writelines
  - with open() as file:
    for line in file
- Function
  - Default arguments
  - Positional and keyword arguments
    - *args, **kwargs
  - Will a Variable's Value Change After a Function Call?
  - Variable scope
- Modules
- Higher-order functions
  - Function can be the value of a variable
  - Can also be passed and returned just like any other reference variables

Outline

- Recursion
- Docstring

Recursion

- In computer science, recursion is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem.[1]
- Recursion occurs when the definition of a concept or process depends on a simpler or previous version of itself.[2]

1. https://en.wikipedia.org/wiki/Recursion_(computer_science)
2. https://en.wikipedia.org/wiki/Recursion
3. https://medium.com/analytics-vidhya/recursion-the-nesting-doll-of-programming-404ae61708a0

Factorial

- $n! = \begin{cases} 1, & n = 0 \\ n * (n-1)!, & n > 0 \end{cases}$
- n=3

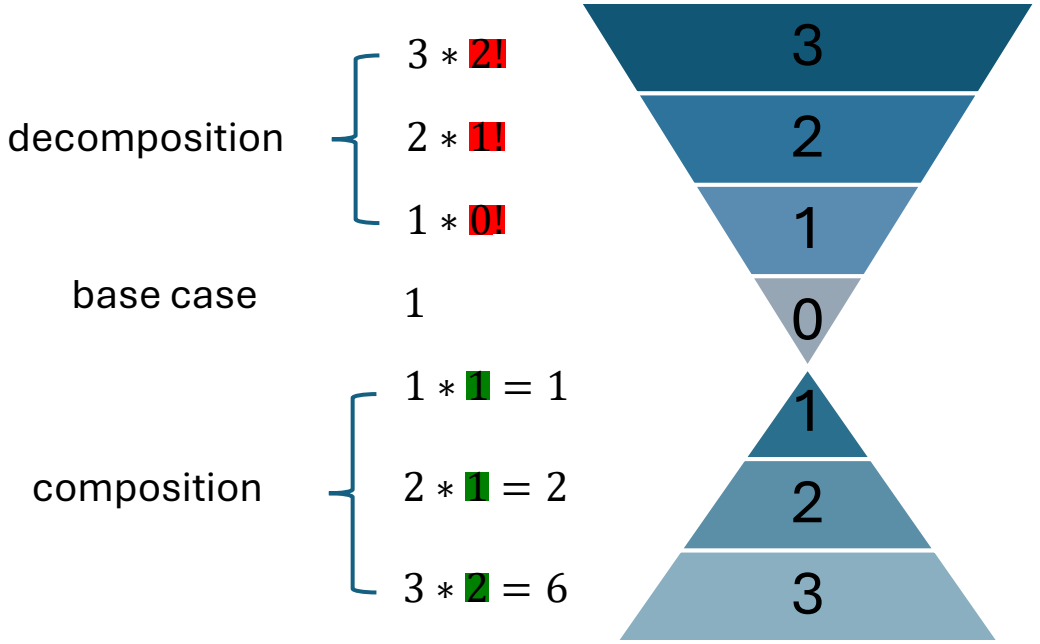
$$n! = 3 * (3-1)! = 3 * 2! = 3 * (2 * (2-1)!) = 3 * (2 * 1!) = 3 * (2 * (1 * (1-1)!)) = 3 * (2 * (1 * 0!)) = 6$$

How to we write a program to solve this problem?

Factorial

- decomposition
  - We turn this problem into a smaller problem of same kind.
- base case
  - Finally, we know the answer without calling the function itself
- composition
  - We use the result of smaller problems to find the result of a larger problem.

$$n! = \begin{cases} 1, & n = 0 \\ n * (n-1)!, & n > 0 \end{cases}$$



Factorial

- https://pythontutor.com/render.html#mode=display
- Further Reading:
  - Stack
    - Last in first out
    - Push and pop
  - Frame
    - To represent the execution state of a function or code block

```
def factorial(n):
    # Base case: If n is 0, return 1
    if n == 0:
        return 1
    # Recursive case: n! = n * (n-1)!
    else:
        return n * factorial(n - 1)
# Example usage
number = int(input("Please enter an integer: "))
result = factorial(number)
print(f"The factorial of {number} is {result}")
```

Key points

- Figure out the formula about how a big problem can be solved by a similar smaller problem
  - Assume you know how to solve the problem for *n-1*. How to use this information to solve the problem for *n*?
  - Figure out what is the base case that the problem can get a solution.

Fibonacci numbers

- Leonardo Pisano Fibonacci (12thcentury) is credited for the sequence:
  - f(n): 0, 1, 1, 2, 3, 5, 8, 13, 21, …
  - n: 0, 1, 2, 3, 4, 5, 6, 7, 8, …
- Starting from the third number, each number in the sequence is the sum of the previous two.
- Calculate f(n) to calculate the number on the position n

Fibonacci numbers

- Figure out the formula about how a big problem can be solved by a similar smaller problem
  - Assume you know how to solve the problem for *n-1*. How to use this information to solve the problem for *n*? Derive the recursive formula.
  - Figure out what is the base case that the problem can get a solution.

$$F_0 = 0$$
- $F_1 = 1$

Base case

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

Recursive formula

- Practice

From analyse to program

```
def func(big_problem):
    if base_case:
        return value
    else:
        # recursive formula
        tmp = func(smaller_problem)
        return recursive_formula(tmp)
```

Fibonacci numbers

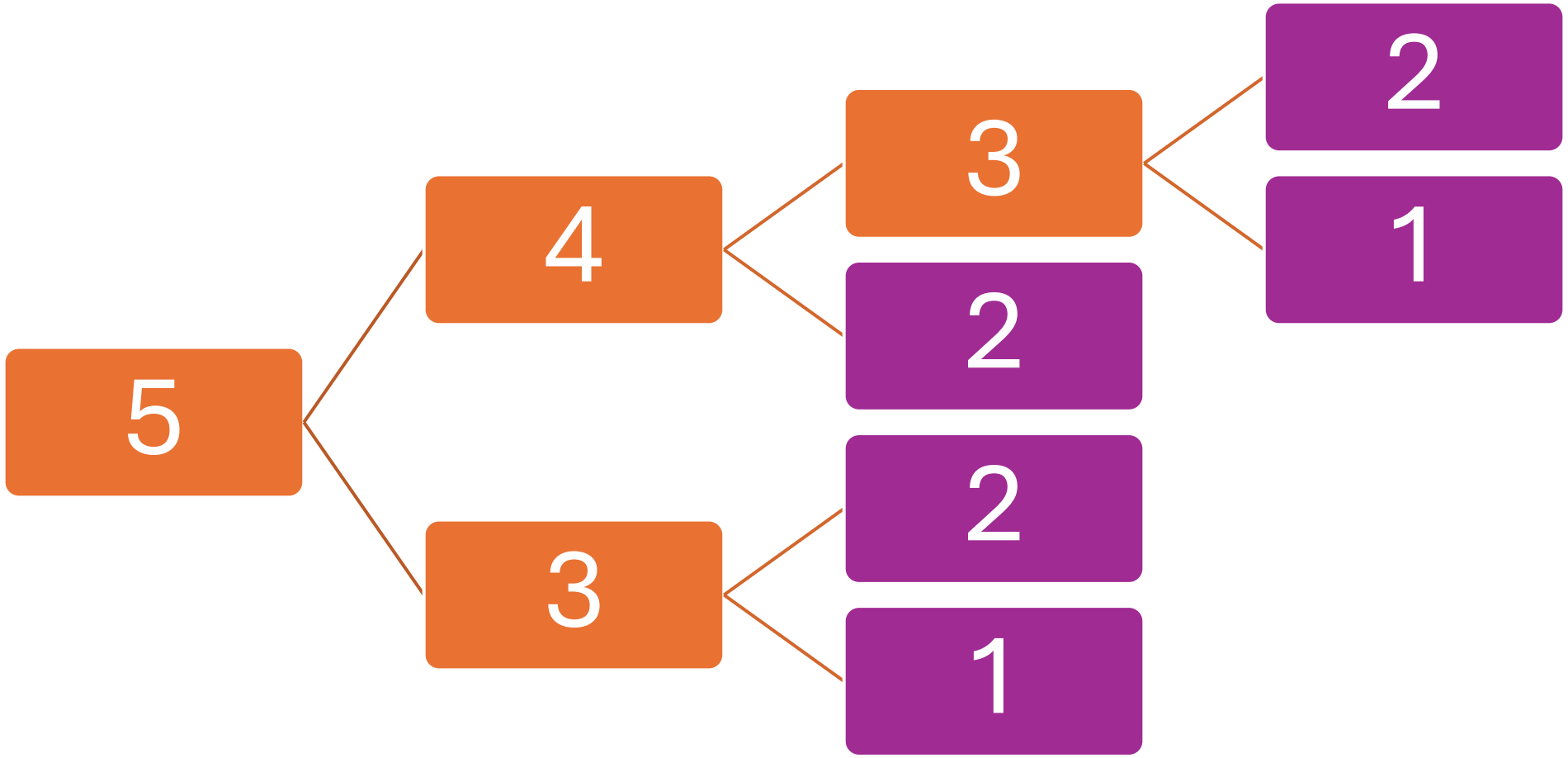
```
0s
def fib(n):
  print(f'begin fib({n})')
  if n==1 or n==2:
    return 1
  else:
    return fib(n-1)+fib(n-2)

a = fib(5)
print(a)
```

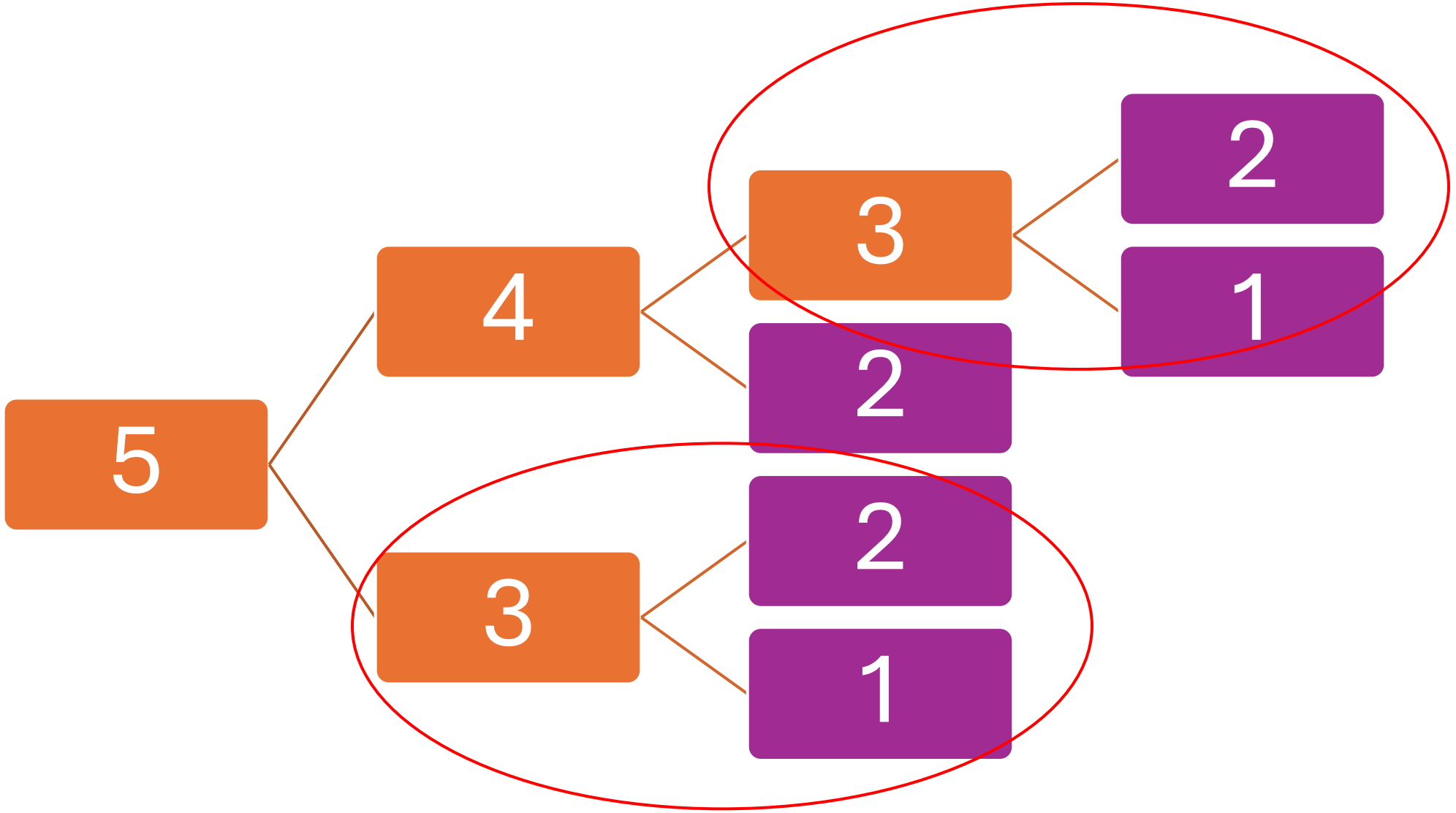
```
begin fib(5)
begin fib(4)
begin fib(3)
begin fib(2)
begin fib(1)
begin fib(2)
begin fib(3)
begin fib(2)
begin fib(1)
5
```

- Anything can be improved?

Fibonacci numbers



Fibonacci numbers



Fibonacci numbers

```python
fib_dict  = {1:1,2:1}
def fib(n):
  print(f'begin fib({n})')
  if n==1 or n==2:
    return 1
  else:
    if n in fib_dict:
      return fib_dict[n]
    else:
      fib_dict[n] = fib(n-1)+fib(n-2)
      return fib_dict[n]

a = fib(5)
print(a)
```

```
begin fib(5)
begin fib(4)
begin fib(3)
begin fib(2)
begin fib(1)
begin fib(2)
begin fib(3)
5
```

Practice

- Reverse a list
  - scores = [70,63,98,85,22]
  - Figure out the formula about how a big problem can be solved by a similar smaller problem
    - Assume you know how to solve the problem for n-1. How to use this information to solve the problem for n?
    - Figure out what is the base case that the problem can get a solution.

  - What is the base case?
  - What is the recursive formula?

https://leetcode.com/problems/reverse-linked-list/

Practice

0s

```
scores = [70,63,98,85,22]
def reverse(scores):
  if len(scores)==0:
    return []
  elif len(scores)==1:
    return scores
  else:
    return reverse(scores[1:])+[scores[0]]

print(reverse(scores))
```

```
[22, 85, 98, 63, 70]
```

Practice

- Implement Factorial function without recursion

$$n! = \begin{cases} 1, & n = 0 \\ n * (n-1)!, & n > 0 \end{cases}$$

- Recursion problems can be written in an iterative manner, and vice versa
- In actual programming, the choice between recursion and iteration often depends on the nature and context of the specific problem, as well as considerations of performance and readability.

Practice

```
def factorial_loop(n):
  res = 1
  for i in range(1,n+1):
    res = res*i
  return res

print(factorial_loop(5))
```

Review

- Recursion
  - Figure out the formula about how a big problem can be solved by a similar smaller problem
  - Base case
- Further Reading
  - Dynamic Programming
  - Stack
  - Frame

Docstring

- A Docstring is a comment that appears as the first line in a new part of the code
- Used for help functions

```
import math
help(math)
```

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

        The result is between 0 and pi.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

        The result is between -pi/2 and pi/2.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.

        The result is between -pi/2 and pi/2.

    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered.
```

Write your own module

- In the beginning of the module, write the module description surrounded by ''' or """
- In each function, write your comment in the first line with ''' or """

txt_processor.py

```
'''
A module that processing text files
'''
def load_data(file_name):
    '''
    load txt lines from a file
    '''
    with open(file_name,'r') as file:
        data = file.readlines()
        return data

def process_data(data):
    '''
    data is a list of string
    for each string, there is a \\n at the end
    '''
    data = [i.strip()+': processed\n' for i in data]
    return data
```

Write your own module

test.py

```
1  import txt_processor
2  help(txt_processor)
3  #from txt_processor import load_data, process_data
```

```
Help on module txt_processor:

NAME
    txt_processor - A module that processing text files

FUNCTIONS
    load_data(file_name)
        load txt lines from a file

    process_data(data)
        data is a list of string
        for each string, there is a \n at the end

FILE
```

Review

- A comment that appears as the first line in a new part of the code
- Surrounded by ''' or """

Byte Data Type

- ASCII, Byte, bit, Unicode
- Beyond txt file: image, audio, video, npy

Byte Data Type

- The byte is a unit of digital information that most commonly consists of eight bits.
- A bit is 0 and 1.
- The bit is the most basic unit of information in computing and digital communication.
- The name bit is a portmanteau of binary digit.
  - The bit represents a logical state with one of two possible values. These values are most commonly represented as either "1" or "0", but other representations such as true/false, yes/no, on/off, or +/− are also widely used.
- What is the value range of a byte?

Memory usage of int and float in Python

- Not fixed
- A surprise from Python
  - import sys
  - print(sys.getsizeof(1.0)) # 24
  - print(sys.getsizeof(1)) # 28
  - reference count, type information, extra padding or management overhead, etc.

Byte and ASCII code

- One byte represents 8 bits
- In python, a char type is one byte
- The original ASCII table is a 7-bit encoding scheme that defines 128 characters, numbered from 0 to 127.
- These characters include English uppercase and lowercase letters, digits, punctuation marks, and control characters.
- The extended ASCII table (characters numbered 128 to 255) varies across different platforms and systems, lacking a single standard version.

ASCII Table

- ASCII stands for American Standard Code for Information Interchange.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

ASCII Table

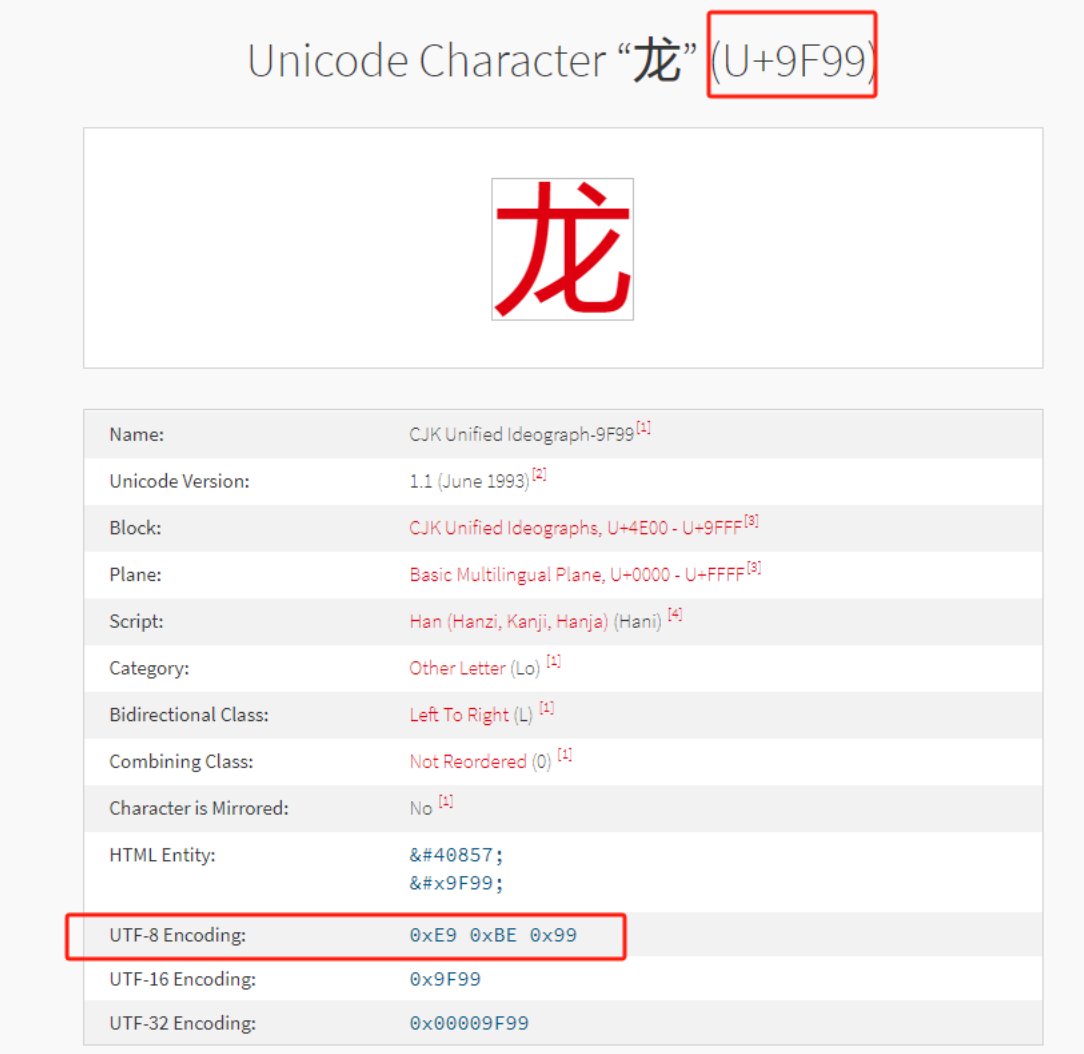
- Extended ASCII Codes
- Has not been widely or consistently adopted

128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

Unicode

- The original ASCII table is a 7-bit encoding scheme that defines 128 characters, numbered from 0 to 127.
- These characters include English uppercase and lowercase letters, digits, punctuation marks, and control characters.
- How to handle other languages? Chinese, Tamil, etc.
  - Too many characters
  - A-65
  - 龙-40857

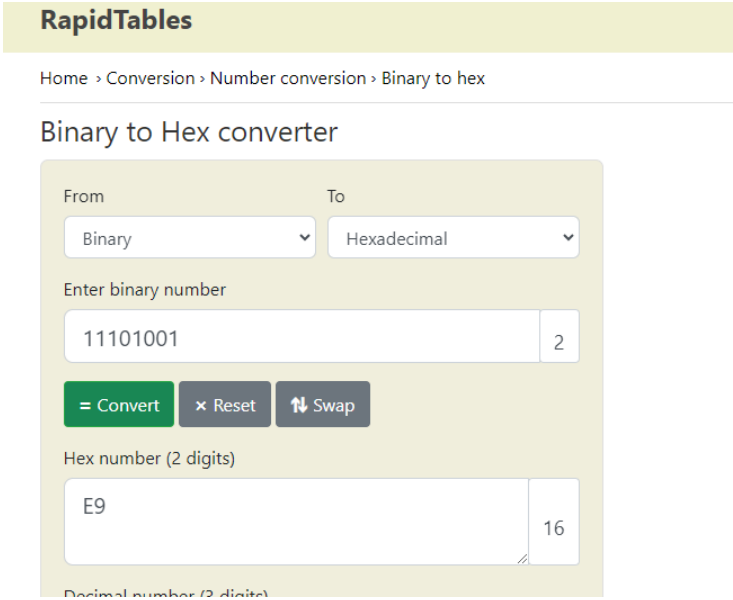


Unicode and UTF-8

- A – 65 – 1000001 – 0100 0001
- 龙 – 40857 – 1001 1111 1001 1001 – (1001 1111) (1001 1001)
- UTF-8
  - Variable-Length Encoding
  - Uses 1 to 4 bytes to encode characters.
  - ASCII characters use 1 byte, while other characters use 2 to 4 bytes.
  - Full compatible with ASCII
    - Any valid ASCII text is also valid UTF-8 text.

Byte and UTF-8

- The character "龙" (U+9F99) is encoded in UTF-8 as E9 BE 99.
- 龙 – 40857 – 1001 1111 1001 1001 - 1001 111110 011001
- UTF-8 uses 3 bytes to represent "龙"
- Format: 1110xxxx 10xxxxxx 10xxxxxx
- First byte 1110xxxx
  - Fill in the first four bits: 1110 1001, which is E9.
- Second byte 10xxxxxx
  - Fill in the next six bits: 1011 1110, which is BE.
- Third byte 10xxxxxx:
  - Fill in the remaining six bits: 1001 1001, which is 99.



https://www.rapidtables.com/convert/number/binary-to-decimal.html

+ Code + Text

```
with open('1.txt','rb') as f_first:
  char = f_first.readline()
  print(char)
print('='*20)
with open('1.txt','r',encoding='utf-8') as f_first:
  char = f_first.readline()
  print(char)
print('='*20)
with open('1.txt','r',encoding='ascii') as f_first:
  char = f_first.readline()
  print(char)
```

```
b'\xe9\xbe\x99'
====================
龙
====================
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
<ipython-input-7-3560149d1c79> in <cell line: 9>()
      8 print('='*20)
      9 with open('1.txt','r',encoding='ascii') as f_first:
---> 10   char = f_first.readline()
     11   print(char)
     12

/usr/lib/python3.10/encodings/ascii.py in decode(self, input, final)
     24 class IncrementalDecoder(codecs.IncrementalDecoder):
     25     def decode(self, input, final=False):
---> 26         return codecs.ascii_decode(input, self.errors)[0]
     27
     28 class StreamWriter(Codec,codecs.StreamWriter):

UnicodeDecodeError: 'ascii' codec can't decode byte 0xe9 in position 0: ordinal not in range(128)
```

Next steps: Explain error

1.txt

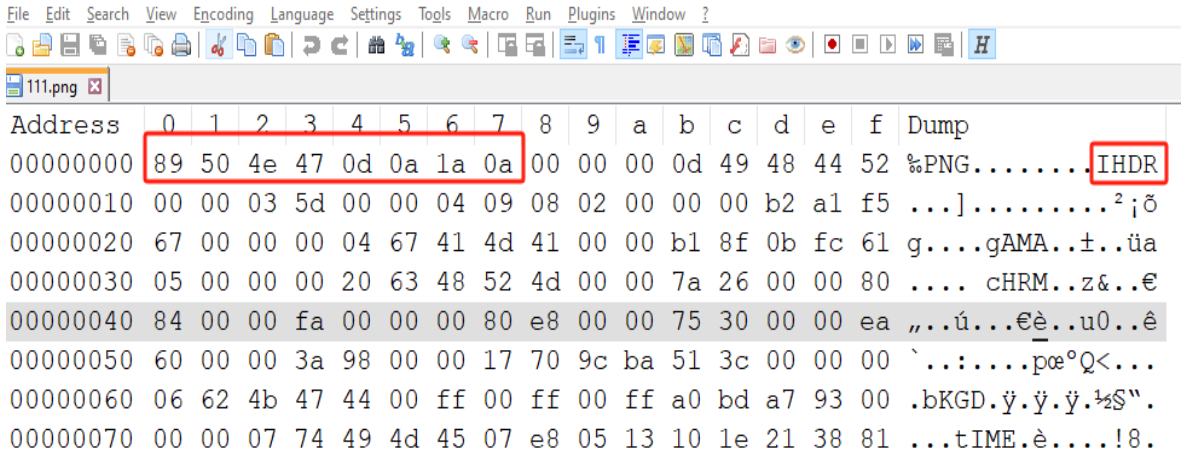
1 龙

UTF-8

Type	Description	Unicode range	Format
1 Byte	Encode 7-bit ASCII characters	U+0000 to U+007F	0xxxxxxx
2 Bytes	Encode 8 to 11-bit characters	U+0080 to U+07FF	110xxxxx 10xxxxxx
3 Bytes	Encode 12- to 16-bit characters	U+0800 to U+FFFF	1110xxxx 10xxxxxx 10xxxxxx
4 Bytes	Encode 17- to 21-bit characters	U+10000 to U+10FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Understanding binary files

- Structure of a PNG File
  - Signature
    - First 8 bytes of the file
    - Specific byte values (hexadecimal)
      - 89 50 4E 47 0D 0A 1A 0A
    - Indicates that the file is a PNG
  - Chunks
    - Critical chunks (must appear in the file)
      - IHDR: Image header
        - Width,Height, Bit depth, Color type, …
      - IDAT: Image data
      - IEND: Image end
    - Ancillary chunks (optional metadata)



Notepad++
Install HexViewer plugin

Bits, Bytes, and Character Encoding

- **Bit**:
  - **1 bit** represents the most basic unit of data in computing, either **0** or **1**.
- **Byte**:
  - A **byte** is the **standard unit** for storing data in computers, comprising **8 bits**.
  - A byte can represent 256 possible values (from 0 to 255 in decimal).
- **ASCII** (American Standard Code for Information Interchange):
  - ASCII uses **7 bits** to encode printable characters, including English letters, digits, and control characters.
  - Typically, ASCII is represented using **8 bits** (1 byte), where the highest bit is often set to 0.
- **UTF-8** (Unicode Transformation Format - 8 bits):
  - UTF-8 is a **variable-length** encoding scheme that uses **1 to 4 bytes** to represent characters from all languages.
  - It is backward compatible with ASCII: ASCII characters use 1 byte, while more complex characters (e.g., from other languages) can use up to 4 bytes.
- **Binary Files**:
  - A **binary file** is any file that contains a sequence of bytes.
  - Not all binary files are **human-readable**. Text files, such as those using **ASCII** or **UTF-8** encoding, can be read by humans. However, binary files like images, audio, or executables often need to be **decoded** by specific programs to be interpreted in a human-readable form.

Professional

UTF-8 won't be on the exam.