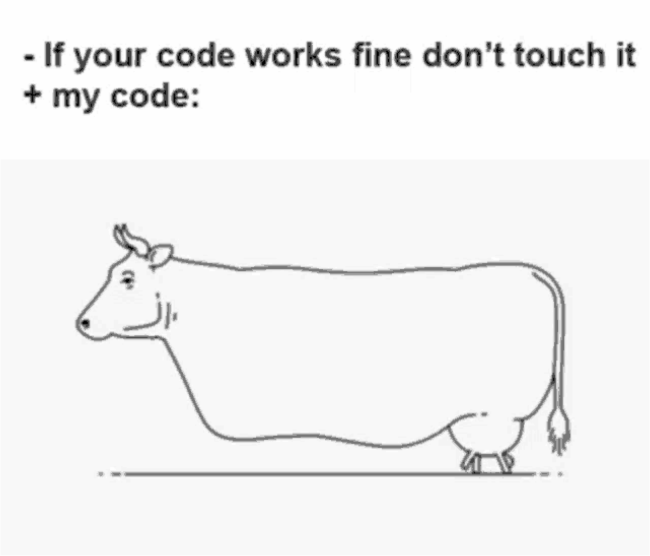
(U4284) Python程式設計
Basic



Speaker: 吳淳硯

Class Contents

- Contents
  1) Basic
  2) Flow Control
  3) Function, Class
  4) Package Intro (NumPy, pandas, seaborn, matplotlib)
  5) Python feat. VBA (pywin32)
  6) Python feat. SQL (pyodbc)
  7) Linear Model (statsmodels)
  8) Linear Program (pulp)

- If time permits
  9) Machine Learning (scikit learn)
  10) Deep Learning (keras, tensorflow)

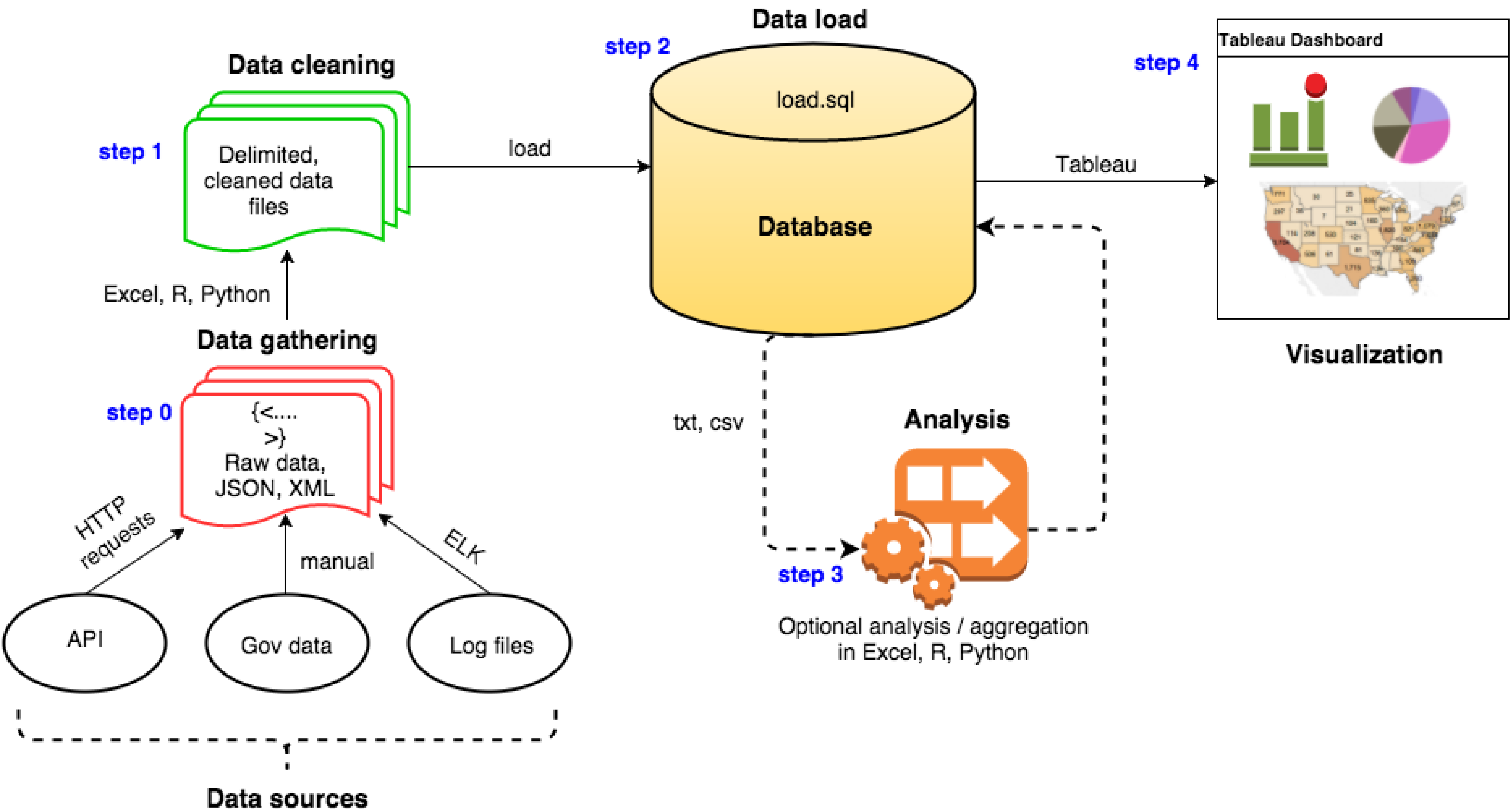
Evaluation & Reference

- Evaluation
  - Class Assignment ($10\% \times 4 = 40\%$)
  - Side Project ($15\% \times 2 = 30\%$)
  - Attendance ($6\% \times 5 = 30\%$)

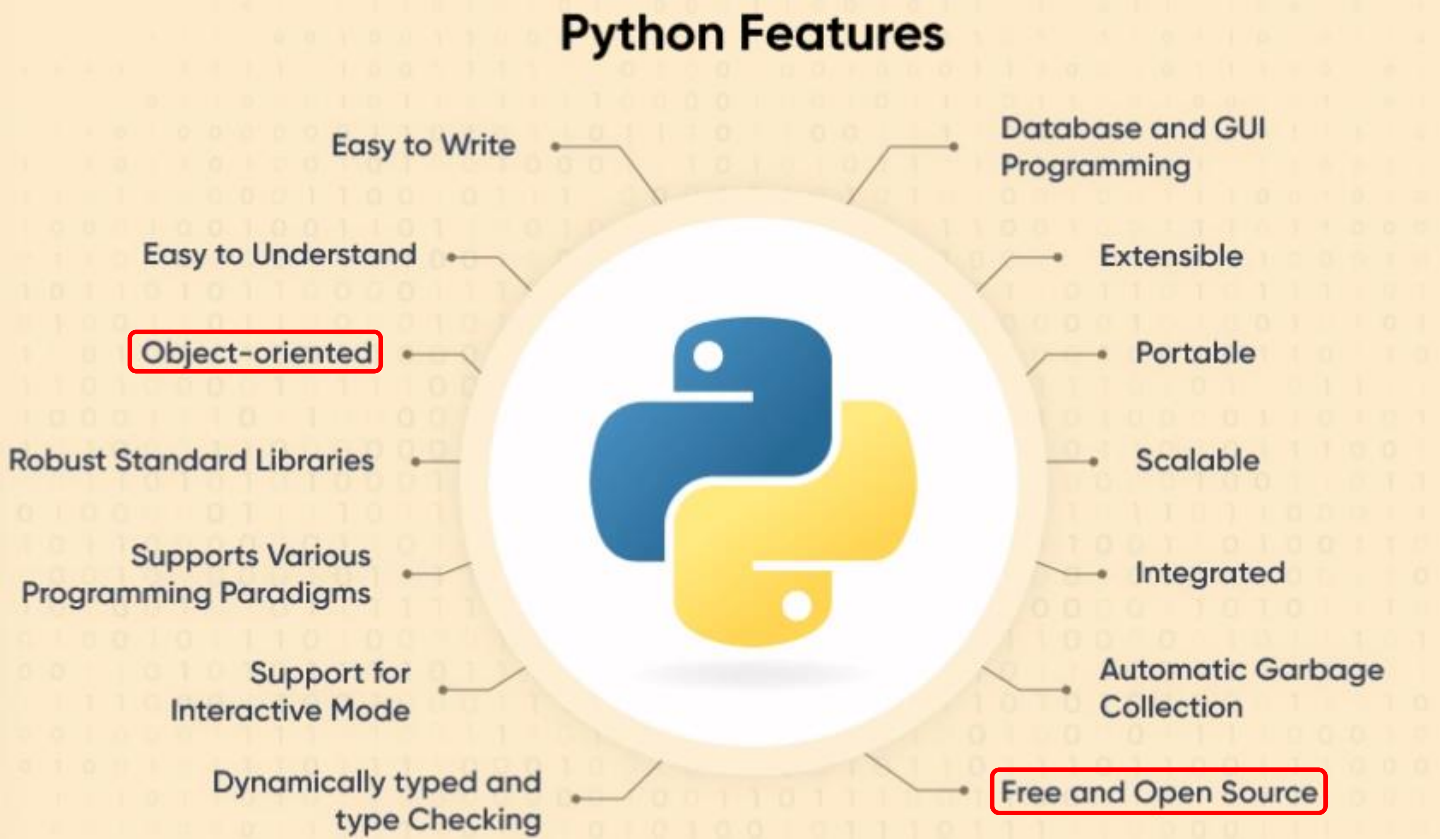
- Teaching Methods:
  - Lecture

- Reference
  - Introduction Python 2ed – Bill Lubanovic
  - Linear Models with Python – Julian Faraway
  - Python Data Science Handbook 2ed – Jake VanderPlas
  - SQL Pocket Guide – Alice Zhao

Data Analysis Workflow

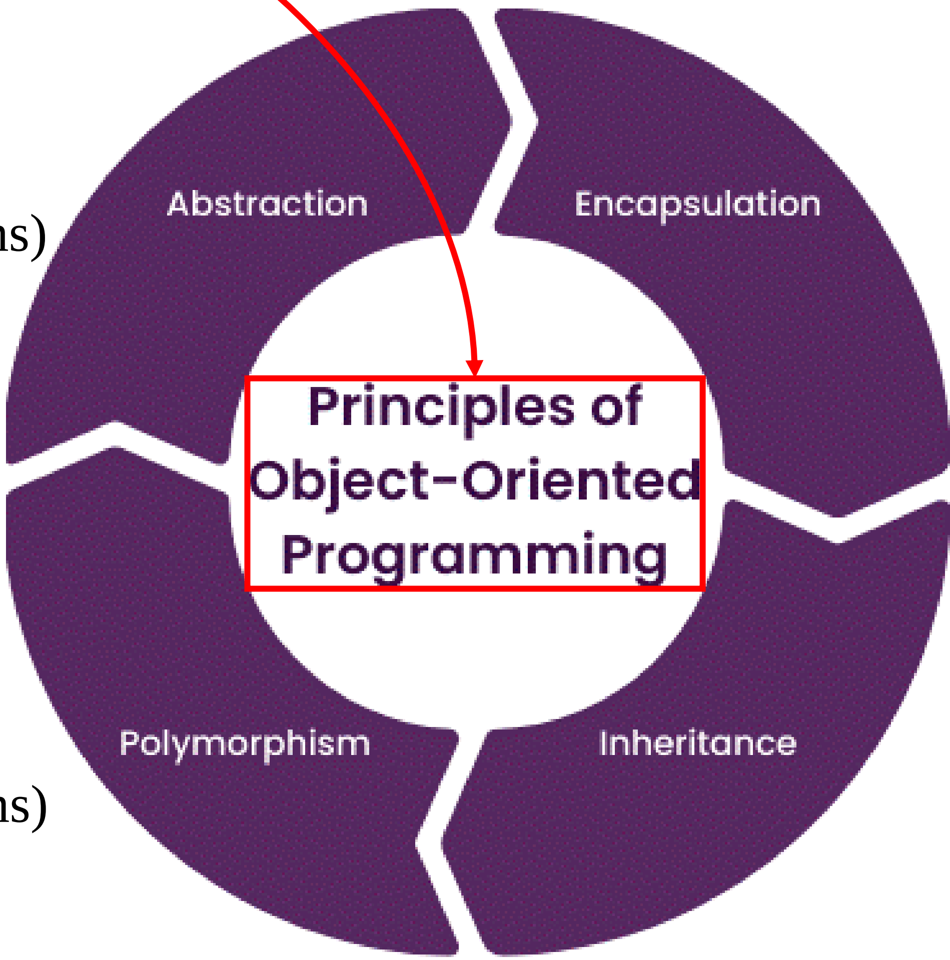


Python


Python Features
Easy to Write
Database and GUI Programming
Easy to Understand
Extensible
Object-oriented
Portable
Robust Standard Libraries
Scalable
Supports Various Programming Paradigms
Integrated
Support for Interactive Mode
Automatic Garbage Collection
Dynamically typed and type Checking
Free and Open Source

Principles of OOP

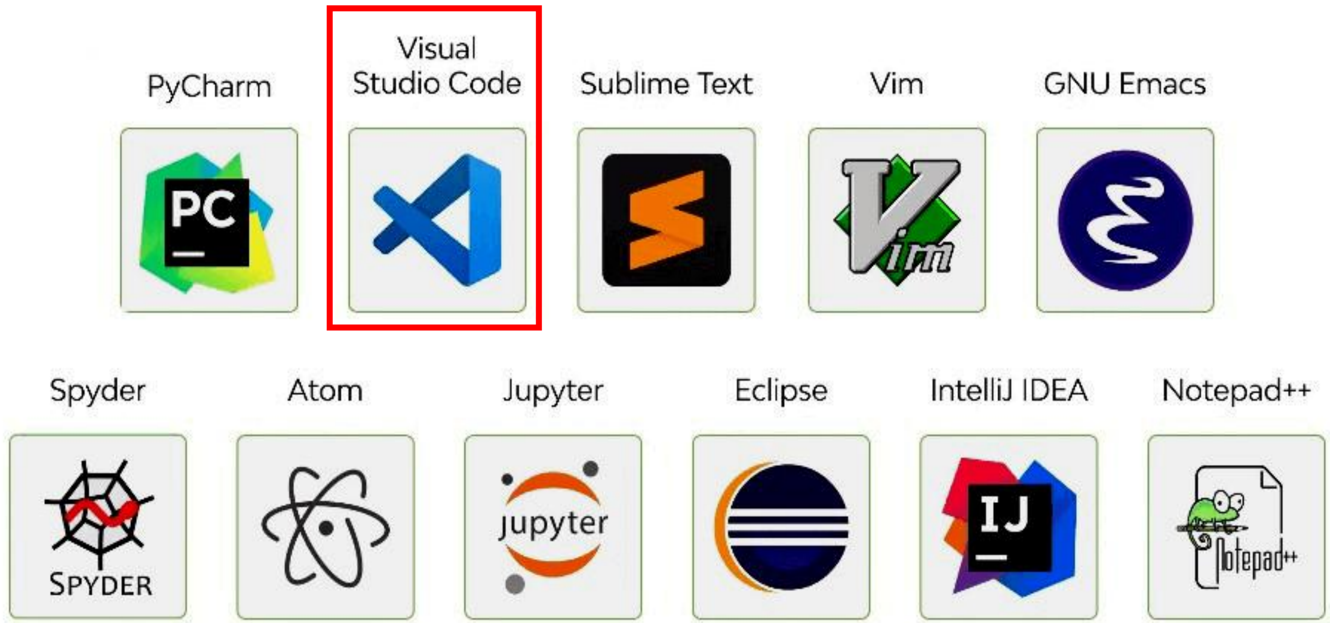
- *Encapsulation* (Data Hiding)
  - Wrap data (variables) and methods (functions) inside a class, keeping some details hidden from the outside.
- *Abstraction* (Simplifying Complex Systems)
  - Show only the necessary details and hide complex logic.
- *Inheritance* (Reusability)
  - A child class can inherit properties and behavior from a parent class.
- *Polymorphism* (One Interface, Many Forms)
  - The same function or method can work differently depending on the object that calls it.



IDE (Integrated Development Environment software)

- IDE is a software application that helps programmers develop software code efficiently.

- It makes coding easier and faster.



Intro

- A object is a chunk of data that contains at least the following
  - A *type* that defines what it can do
  - A unique *id* to distinguish it from other objects
  - A *value* consistent with its type
  - A *reference* count that tracks how often this object is used
- Variable names have some rules
  - Lowercase letters [a-z]
  - Uppercase letters [A-Z]
  - Digits [0-9]
  - Underscore
- They must begin with a letter or an underscore, not a digit.
- Names that begin with an underscore are treated specially.
- They can't be one of Python's reserved words (keywords).

Data type

Table 2-1. Python's basic data types

Name	Type	Mutable?	Examples	Chapter
Boolean	`bool`	no	`True`, `False`	Chapter 3
Integer	`int`	no	`47`, `25000`, `25_000`	Chapter 3
Floating point	`float`	no	`3.14`, `2.7e5`	Chapter 3
Complex	`complex`	no	`3j`, `5 + 9j`	Chapter 22
Text string	`str`	no	`'alas'`, `"alack"`, `'''a verse attack'''`	Chapter 5
List	`list`	yes	`['Winken', 'Blinken', 'Nod']`	Chapter 7
Tuple	`tuple`	no	`(2, 4, 8)`	Chapter 7
Bytes	`bytes`	no	`b'ab\xff'`	Chapter 12
ByteArray	`bytearray`	yes	`bytearray(...)`	Chapter 12
Set	`set`	yes	`set([3, 5, 7])`	Chapter 8
Frozen set	`frozenset`	no	`frozenset(['Elsa', 'Otto'])`	Chapter 8
Dictionary	`dict`	yes	`{'game': 'bingo', 'dog': 'dingo', 'drummer': 'Ringo'}`	Chapter 8

Mutability

- The type also determines whether the data value contained by the box can be changed (mutable) or constant (immutable).

- A mutable object is like a box with a lid:
  - Not only can you see the value inside, you can also change it.

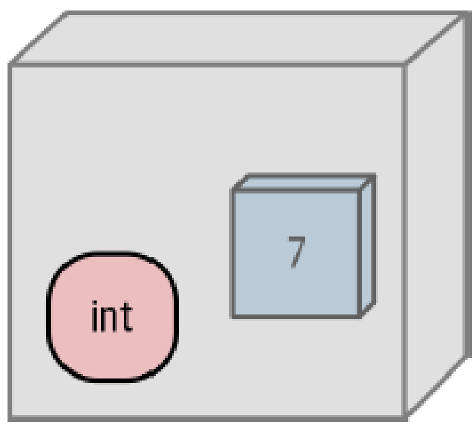


Figure 2-1. An object is like a box; this one is an integer with value 7

Comment String

- A *comment* is a piece of text in your program that is ignored by Python interpreter.

- You mark a comment by using the # character; everything from that point on the end of the current line is part if the comment.

These are valid names:

- `a`
- `a1`
- `a_b_c___95`
- `_abc`
- `_1a`

These names, however, are not valid:

- `1`
- `1a`
- `1_`
- `name!`
- `another-name`

Assignment

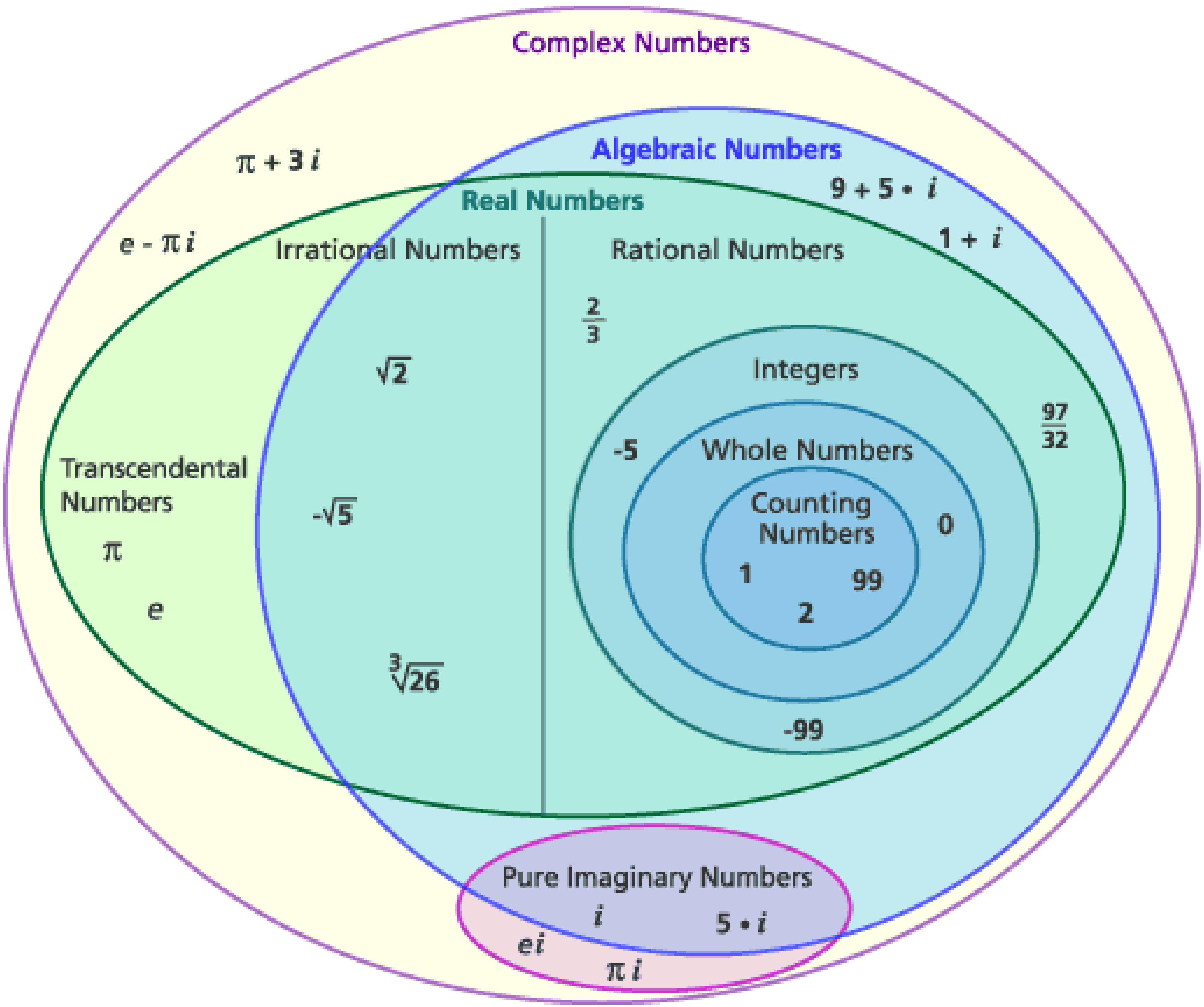
- Computer program lines may look like equations, but their meaning is different. In Python, use = to ***assign*** a value to a variable.

- In math, = means ***equality*** of both sides, but programs it means ***assignment*** (assign the **value on the right side** to the **variable on the left side**).

- In programs, everything on the right side needs to have a value (this is called being ***initialized***). The right side can be a literal value, or a variable that has already been assigned a value, or a combination.

> We all learned in grade school arithmetic that = means *equal to*. So why do many computer languages, including Python, use = for assignment? One reason is that standard keyboards lack logical alternatives such as a left arrow key, and = didn't seem too confusing. Also, in computer programs you use assignment much more than you test for equality.

Number systems



Numbers

- Booleans – True or False.
- Integers – No fractions, no decimal points. ($\mathbb{Z} = \{\cdots, -1,0,1,\cdots\}$)
- Floats – Numbers with decimal points.

Boolean algebra

- Boolean algebra is a set *A* equipped with
  - Two binary operations ∨ (meet | and) and ∧ (join | or)
  - A unary operation ~ (complement | not)
  - Two elements 0 and 1 in *A*

- For all elements *a*, *b* and *c* of A, the following axioms hold.

$a \vee (b \vee c) = (a \vee b) \vee c$	$a \wedge (b \wedge c) = (a \wedge b) \wedge c$	associativity
$a \vee b = b \vee a$	$a \wedge b = b \wedge a$	commutativity
$a \vee (a \wedge b) = a$	$a \wedge (a \vee b) = a$	absorption
$a \vee 0 = a$	$a \wedge 1 = a$	identity
$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$	$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	distributivity
$a \vee \neg a = 1$	$a \wedge \neg a = 0$	complements

Comparison Operators

- Note that two equal signs (==) are used to *test equality*; remember, a single equal sign (=) is what you use to *assign* a value to a variable.

Python's *comparison operators:*

equality	==
inequality	!=
less than	<
less than or equal	<=
greater than	>
greater than or equal	>=

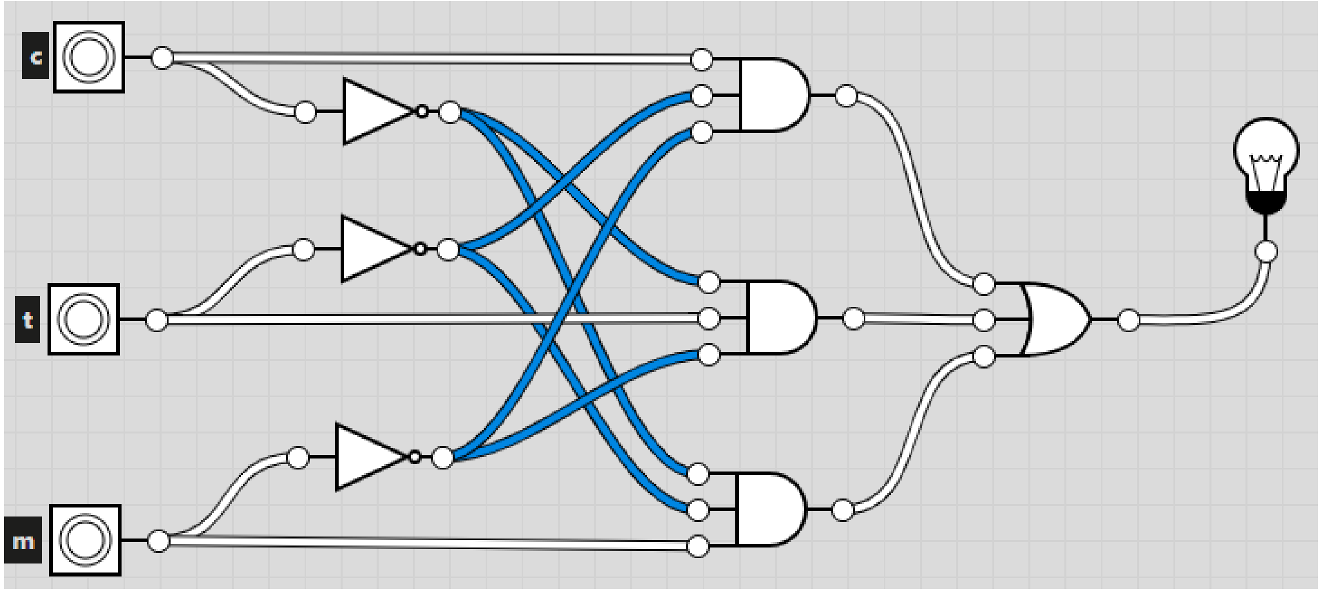
Symmetric Difference

$$A \,\Delta\, B = (A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B)$$

Name	Symbol	Usage	What it does
Bitwise And	&	a&b	Returns 1 Only if both the bits are 1
Bitwise Or	|	a|b	Returns 1 if one of the bits is 1
Bitwise Not	~	~a	Returns the complement of a bit
Bitwise Xor	^	a^b	Returns 0 if both the bits are same else 1

Coffee, Tea or Milk?

- Input
  - c = coffee button (1 = pushed, 0 = not pushed)
  - t = tea button (1 = pushed, 0 = not pushed)
  - m = milk button (1 = pushed, 0 = not pushed)
- Output
  - x = choice verifier (1 = acceptable input / deliver the choice, 0 = unacceptable input / light an error light)
    = c(~t)(~m) + (~c)t(~m) + (~c)(~t)m



c	t	m	x
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1
0	0	0	0

Exercise – U.S. Rocket Launcher

The missile system is to be activated by a device obeying these rules: each member of the Security Council has a button to push; the missiles fire only if the President and at least one Counselor push their buttons.

- Input
  - p = President's button (1 = pushed, 0 = not pushed)
  - x, y, z = Counselors' button (1 = pushed, 0 = not pushed)

- Output
  - f = fire missiles command (1 = fire, 0 = don't fire)

☺Question ?

a. Write the Boolean expression.
b. Draw the circuit.
c. Construct the truth table listing all possibilities

國立臺北大學 National Taipei University

Integers

	Addition	Multiplication
Closure:	$a+b$ is an integer	$a \times b$ is an integer
Associativity:	$a+(b+c)=(a+b)+c$	$a \times (b \times c)=(a \times b) \times c$
Commutativity:	$a+b=b+a$	$a \times b=b \times a$
Existence of an identity element:	$a+0=a$	$a \times 1=a$
Existence of inverse elements:	$a+(-a)=0$	The only invertible integers (called units) are −1 and 1.
Distributivity:	$a \times (b+c)=(a \times b)+(a \times c)$ and $(a+b) \times c=(a \times c)+(b \times c)$	
No zero divisors:		If $a \times b=0$, then $a=0$ or $b=0$ (or both)

$\mathbb{Z}$ under addition is a abelian group

Operator Precedence

Operator	Description and examples
[*v*, ...], {*v1*, ...}, {*k1*: *v1*, ...}, (...)	List/set/dict/generator creation or comprehension, parenthesized expression
seq[*n*], *seq*[*n*:*m*], *func*(*args*...), *obj*.*attr*	Index, slice, function call, attribute reference
**	Exponentiation
+*n*, −*n*, ~*n*	Positive, negative, bitwise not
*, /, //, %	Multiplication, float division, int division, remainder
+, -	Addition, subtraction
<<, >>	Bitwise left, right shifts
&	Bitwise and
|	Bitwise or
in, not in, is, is not, <, <=, >, >=, !=, ==	Membership and equality tests
not *x*	Boolean (logical) not
and	Boolean and
or	Boolean or
if ... else	Conditional expression
lambda ...	lambda expression

Positional Number System

- In positional number systems, a number is represented by a string of digits where position of each digits is associated with a weight. In general, is expressed as
$$d_{m-1}d_{m-2}\cdots d_1 d_0.d_{-1}d_{-2}\cdots d_{-n}$$
where d_{m-1} is referred to as the most significant digit (MSD) and d_{-n} as the least significant digit (LSD).

- Integers are assumed to be decimal (base 10) unless you use prefix to specify another base.

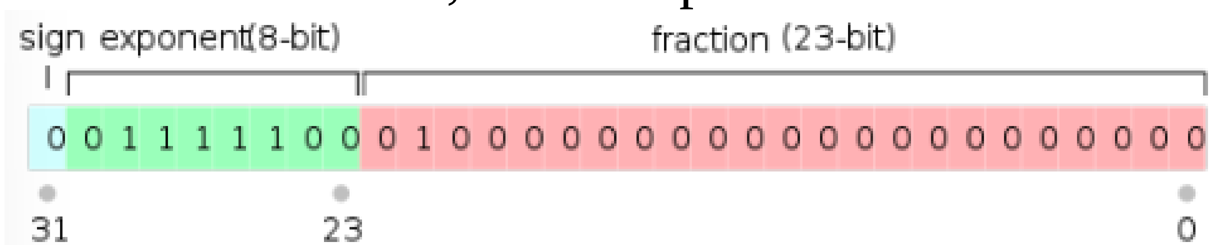
- In Python, you can express literal integers in three bases beside decimal with these integer prefixes:
  - 0b or 0B for *binary* (base 2)
  - 0o or 0O for *octal* (base 8)
  - 0x or 0X for *hex* (base 16)

Floating point representation

- A floating point number in base (b)

$$x = (-1)^s m b^e$$

  consists of three components:
  - the sign ($+$ ($s = 0$) or $-$ ($s = 1$))
  - the mantissa (m)
  - the exponent (e)
- IEEE standard formats :
  - single : 23 bit fraction, 8 bit exponent
  - double : 52 bit fraction, 11 bit exponent



$$\begin{aligned}(-1)^s(1.f)_2 2^{e-127} &= (-1)^0(1.01)_2 2^{01111100_2-127}\\ &= (1+1\times 2^{-2})2^{64+32+16+8+4-127}\\ &= (1.25)2^{-3} = 0.15625\end{aligned}$$

What is Regular Expressions(RE)?

- String containing a combination of *normal characters* and special *metacharacters* that describes patterns to find text or position within a text.

```
r'st\d\s\w{3,10}'
```

- The re module
  - re.findall – return a list of all non-overlapping matches, if any.
  - re.split – split *source* at matches with *pattern* and return a list of the string pieces.
  - re.sub – take another *replacement* argument, and changes all parts of *source* that are matched by *pattern* to *replacement*.
  - re.search – return the first match, if any.
  - re.match – return exact beginning match, if any.

Metacharacters

<table>
<tr><th>Character</th><th>Description</th><th>Example</th></tr>
<tr><td>[]</td><td>A set of characters</td><td>[a-m]</td></tr>
<tr><td>\</td><td>Signals a special sequence</td><td>\d</td></tr>
<tr><td>.</td><td>Any character</td><td>he..o</td></tr>
<tr><td>^</td><td>Starts with</td><td>^hello</td></tr>
<tr><td>$</td><td>Ends with</td><td>world$</td></tr>
<tr><td>*</td><td>0 or more occurrences</td><td>aix*</td></tr>
<tr><td>+</td><td>1 or more occurrences</td><td>aix+</td></tr>
<tr><td>?</td><td>0 or 1 occurrences</td><td>aix?</td></tr>
<tr><td>{m,n}</td><td>Exactly the specified No. of occurrences</td><td>ak{2},ak{3,6}</td></tr>
<tr><td>|</td><td>Either or</td><td>call|put</td></tr>
<tr><td>()</td><td>Capture and group</td><td></td></tr>
</table>

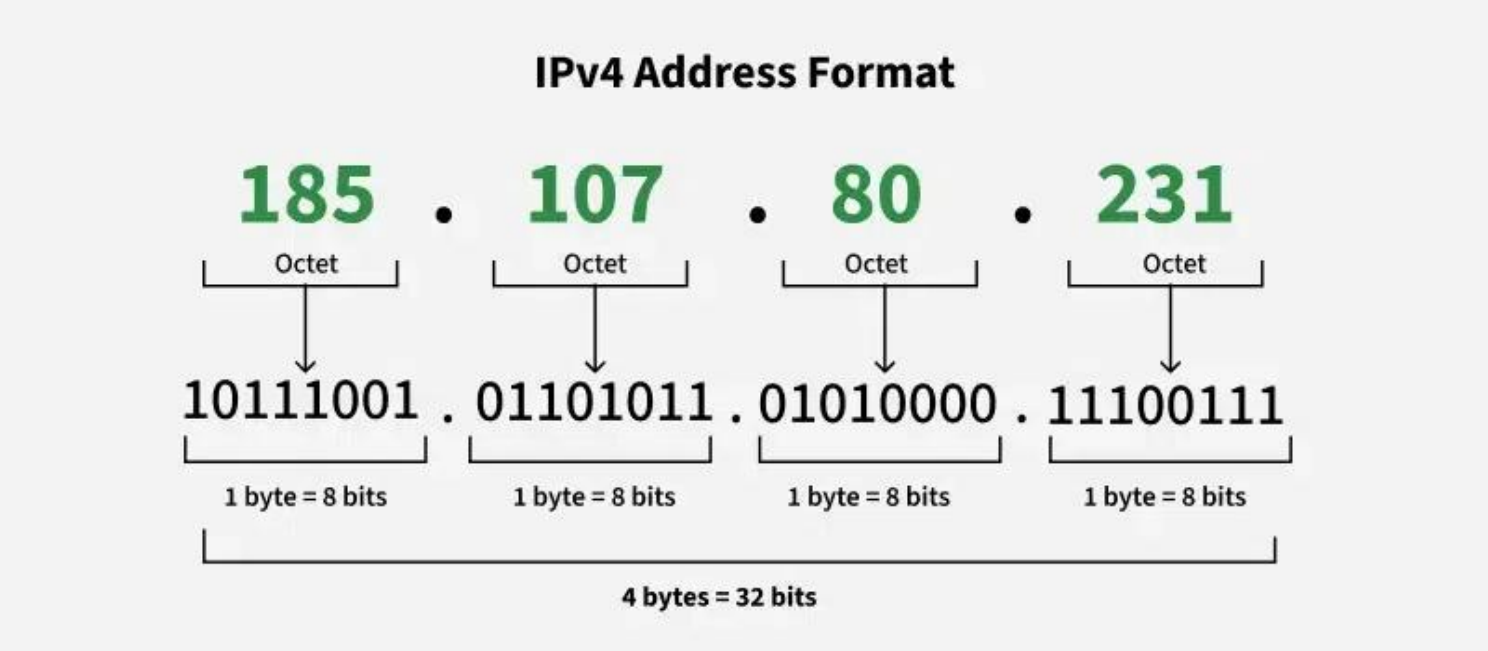
Special Sequences

Character	Description
\A	Returns a match if the specified characters are at the start of the string
\b	Returns a match where the specified characters are at the start or at the end of a word
\B	Returns a match where the specified characters are present, but not at the start (or at the end) of a word
\d	Returns a match where the string contains digits
\D	Returns a match where the string does not contain digits
\s	Returns a match where the string contains a white space character
\S	Returns a match where the string does not contain a white space character
\w	Returns a match where the string contains any word characters
\W	Returns a match where the string does not contain any word characters
\Z	Returns a match if the specified characters are at the end of the string

Exercise – IP v4 & Mobile Number

- What is an IPv4 Address?
  - IPv4 address is a 32-bit numeric address written as 4 decimal numbers (called octets) separated by periods

☺Write the regular expression of detect IP Address.

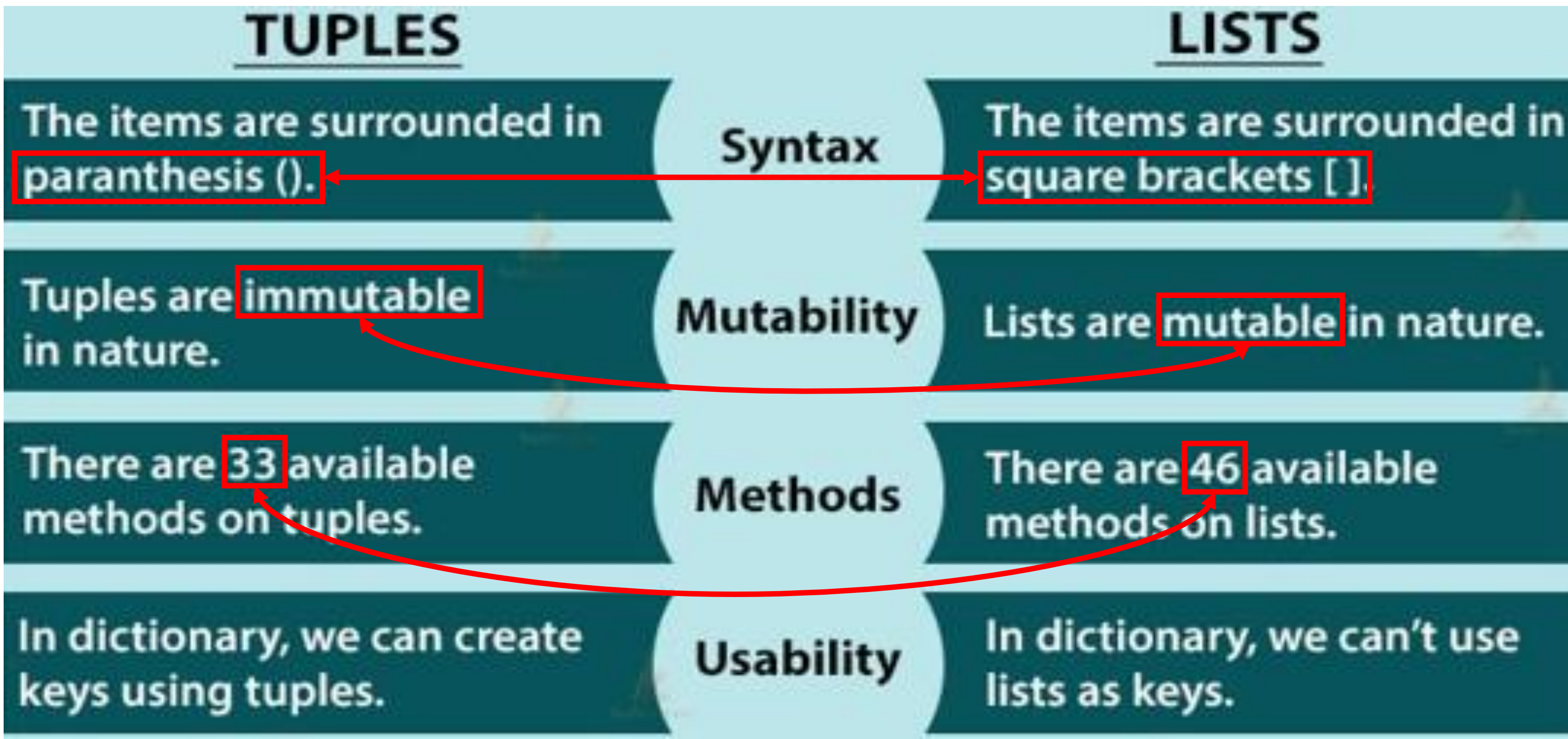


- Taiwan Mobile phone numbers
  - begin in three digits ranging 090~098 with a total length of 10 digits.

☺Write the regular expression of detect Taiwan Mobile number.

Tuples and Lists

- The elements can be of different types. In fact, each element can be any Python object.
- Why does Python contain both lists and tuples?
  - *Mutability*



	TUPLES		LISTS
	The items are surrounded in paranthesis ().	Syntax	The items are surrounded in square brackets [ ].
	Tuples are immutable in nature.	Mutability	Lists are mutable in nature.
	There are 33 available methods on tuples.	Methods	There are 46 available methods on lists.
	In dictionary, we can create keys using tuples.	Usability	In dictionary, we can't use lists as keys.

Dictionaries & Sets

- A *dictionary* is similar to a list, but the order of items does not matter, and they are not selected by an offset such as 0 or 1. Instead, you specify a unique *key* to associate with each *value*.
  - Create with {} or dict()
  - The key can actually be any of Python's immutable types.
  - Copy everything with deepcopy()
  - Dictionary comprehension

```
{key_expression : value_expression for expression in iterable}
```

- A *set* is like a dictionary with its values thrown away, leaving only the keys.
  - Create with {} or set()
  - Create an Immutable set with frozenset()
  - Set comprehension

```
{ expression for expression in iterable }
```

Summary

	Lists	Tuples	Sets	Dictionaries
Ordering	Ordered	Ordered	Unordered	Ordered Unordered before Python 3.7
Indexing	Indexed	Indexed	Not Indexed	Keyed
Mutability	Mutable	Immutable	Mutable Only Adding and Removing	Mutable
Duplicates Allowed	Yes	Yes	No	Yes Only in values and not in keys
Types Allowed	Mutable and Immutable	Mutable and Immutable	Only Immutable	Only Immutable In keys

Generator

- A generator is a Python sequence creation object. With it, you can iterate through potentially huge sequences without creating and storing the entire sequence in memory at once.

- Every time you iterate through a generator, it keeps track of where it was the last time it was called and returns the next value.

- It is a normal function, but it returns its value with a yield statement rather than return.

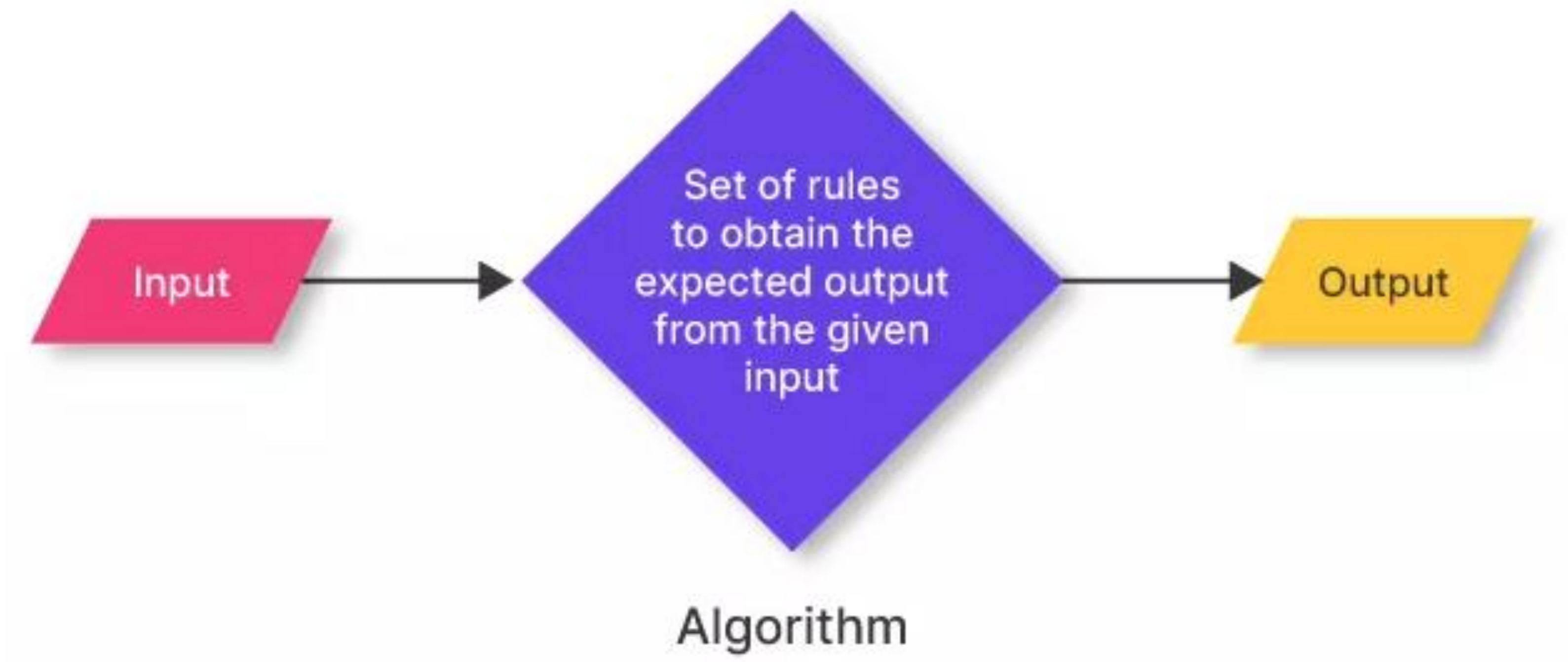
- Generator Comprehension

Definition:

The syntax (<expression> for <var> in <iterable> [if <condition>]) specifies the general form for a **generator comprehension**. This produces a generator, whose instructions for generating its members are provided within the parenthetical statement.

What is an Algorithm?

- A finite set of instructions that accomplishes a particular task. In addition all algorithms must satisfy the following criteria:
  - Input (Zero or more inputs)
  - Output (At least one output)
  - Finiteness (N number of steps)
  - Definiteness (Clear algorithm step)
  - Effectiveness (A carried out step)



What is a good algorithm?

- A good algorithm implemented on a slow computer may perform much better than a bad algorithm implemented on a fast computer.

- What makes an algorithm good?
  - Good time complexity (maybe space complexity)
  - Better than any other algorithm
  - Easy to understand

- How could we measure how much work an algorithm does?
  - Code it and time it. Issues?
  - Count how many instructions it does before implementing it
  - Computer scientists count basic operations, and use a rough measure of this: asymptotic notation

Performance Measurement

- Complexity
  - Space complexity:
    the amount of memory it needs to run to completion.
  - Time complexity:
    the amount of computer (CPU) time it needs to run to completion.
- [TSP problem]
  Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city ?
  Let n = No. of cities, number of possible solution is
  $$(n-1)\times\cdots\times 2\times 1=(n-1)!$$
  $(n-1)!$ Grows very quickly as n grows
  $$n=3\Rightarrow(n-1)!=2$$
  $$n=10\Rightarrow(n-1)!=3628800$$
  assume computer checks 3.15×10^{13} solutions every year,
  $$n=26\approx 5\times 10^{11}\text{years}$$

Asymptotic notations

- [Def - Big O] (*worst case enumeration*)

$$f(n) = O\big(g(n)\big)$$

  if $\exists M, N$ such that

$$|f(n)| \leq M|g(n)| \; \forall n \geq N$$

  Intuitively, this means that f does not grow faster than g.

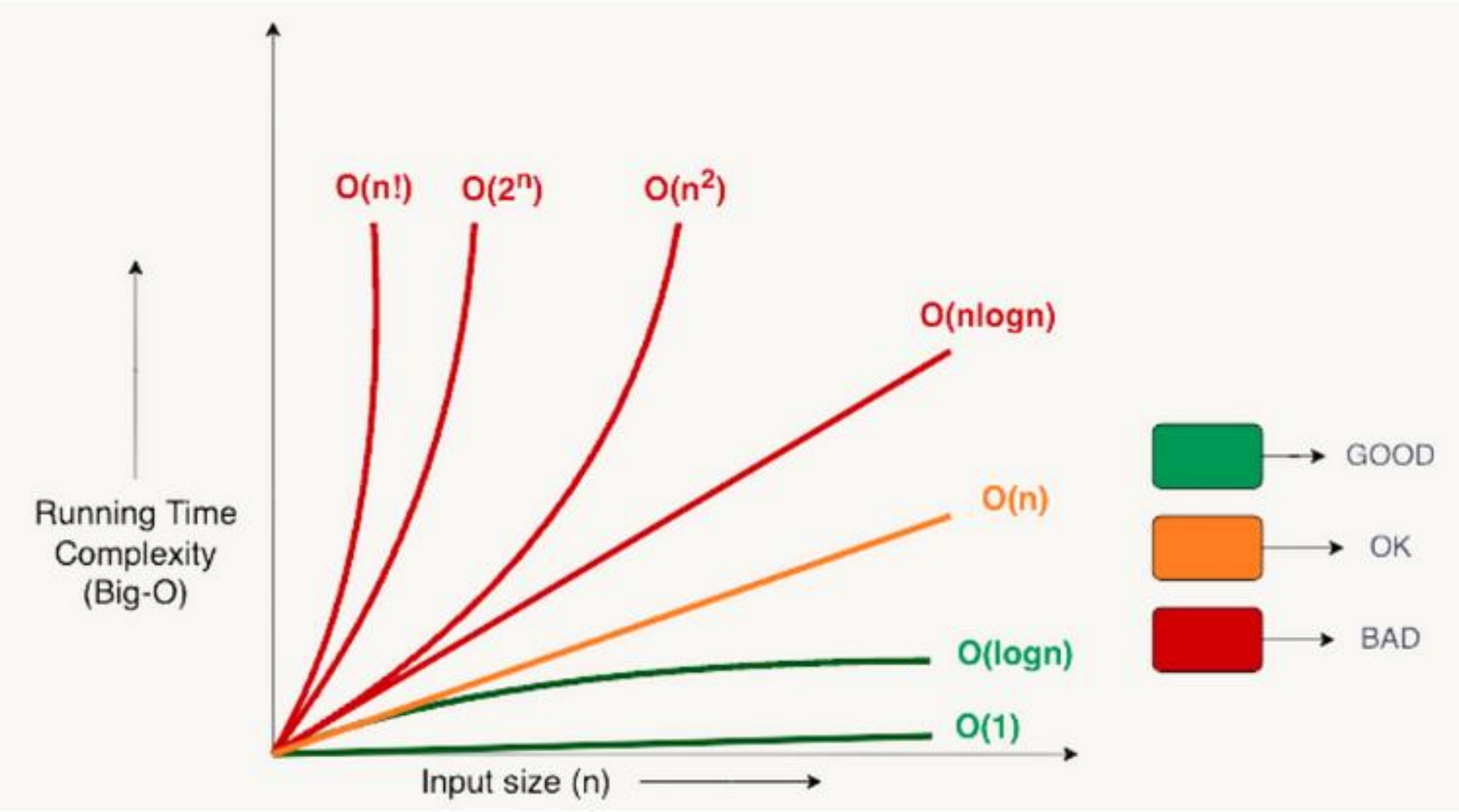
- Classes of functions that are commonly encountered when analyzing algorithms.

Notation	Meaning
$O(\log n)$	Logarithm
$O(n)$	Linear
$O(n \log n)$	Linearithmic
$O(n^2)$	Quadratic
$O(2^n)$	Exponential
$O(n!)$	factorial

Polynomial Time

Non-Polynomial Time

Big O Complexity



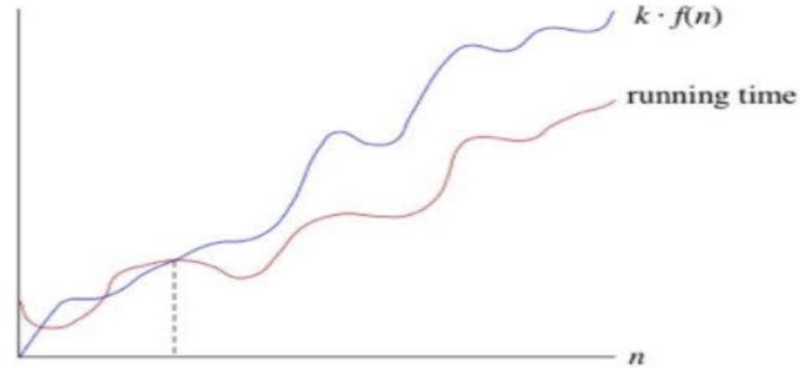
Asymptotic notations

- [Def - *Omega*] (Lower bound Enumeration)

$$f(n) = \Omega\big(g(n)\big)$$

  if $\exists m, N$ such that

$$|f(n)| \geq m g(n) \ \forall n \geq N$$

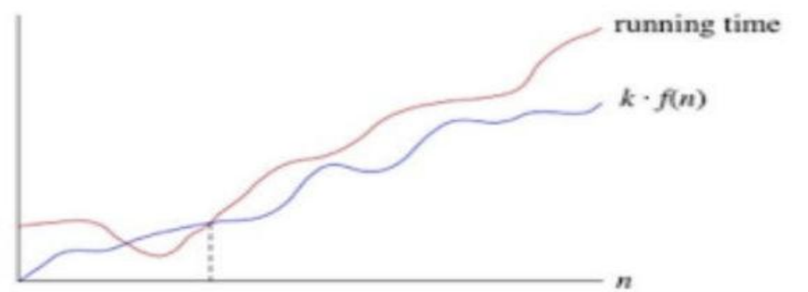


- [Def - *Theta*] (Optimal Enumeration)

$$f(n) = \Theta\big(g(n)\big)$$

  if $\exists m, M, N$ such that

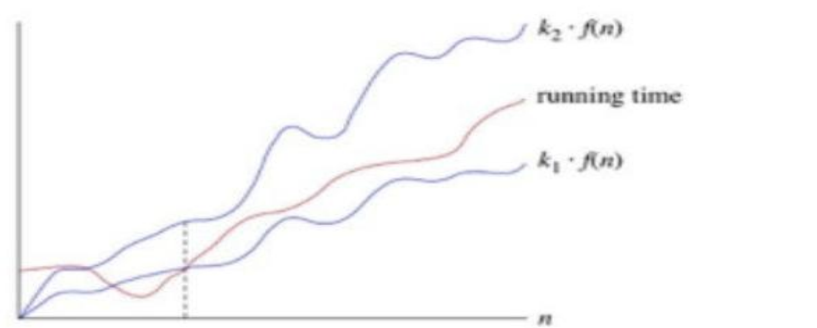
$$m g(n) \leq f(n) \leq M g(n) \ \forall n \geq N$$



- Equivalence

$$f(n) = \Theta\big(g(n)\big)$$

$$\Leftrightarrow f(n) = O\big(g(n)\big) \text{ and } \Omega\big(g(n)\big)$$

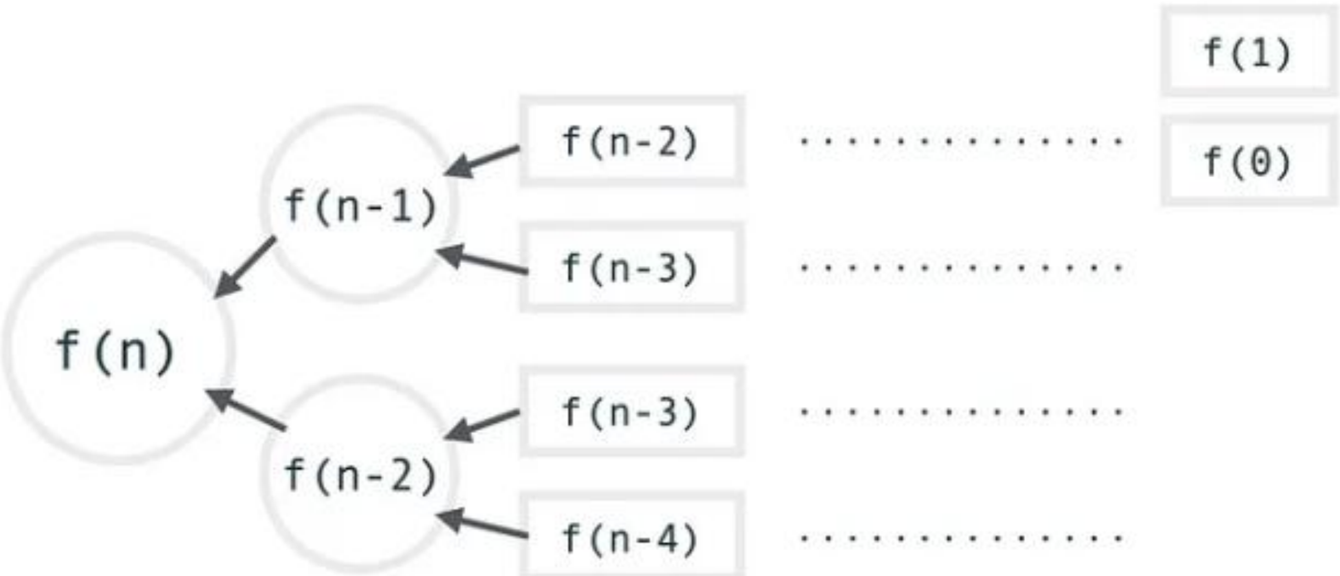


Space Complexity

- Recall Fibonacci number

$$f(n) = f(n-1) + f(n-2)$$
$$f(0) = f(1) = 1$$

- Space Complexity is $O(N)$
  - As in recursion, the space required is proportional to the maximum depth of the recursion tree, and it is N.



Measure the difficulty of the problems

Class	Description
P	problems that can be solved in $\boxed{O(p(n))}$
NP	problems that can be verified in $O(p(n))$
NP-Hard	problems that are at least as hard as all NP problems
$\boxed{\text{NP-Complete}}$	problems in both NP and NP-Hard

Polynomial Time

TSP problem belongs to the NP-Complete