CS1302 Final Review

Final exam

- **Time:** 02-May-2026, 9:30—11:30 Hong Kong time
- **Venue:** On-campus
- **Coverage:** all the required content in topics 1-9
- **Question type:** 10-15 coding questions

Introduction to Computer Programming

- What is computer
- What is programming
- Programming language

Python Basics

Basic Data types

- integers

  int
- floating point numbers

  float
- strings

  str

Variables

- Used to store value—give data a name, so we don't need to remember the data, we only need to remember his name

  x=0

  y=1.02

  s="hello world"

- The name of variables should follow some rules
  - Must start with a letter or _ (an underscore) followed by letters, digits, or _.
  - Must not be a keyword (identifier reserved by Python):

Input and output

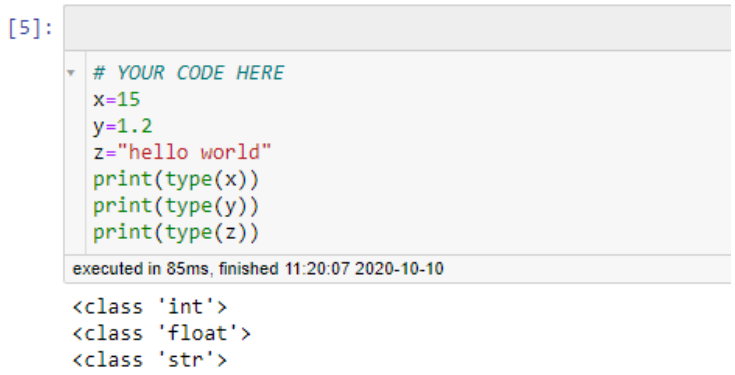
- Use input() to receive user's input
  - input('Please input your name: ')

- Use print() to print out the output
  - print('my name is Jack')
  - there're many parameters, e.g., print('a','b','c',sep='\n',end=' ')
- Can be used together
  - print('my name is', input('Please input your name: '))

Data type conversion

- Sometimes, we need to change data from one data type to another
  - Convert float or string to int: int()
    - The input must be valid, e.g., int('a') is invalid, but int('1') is valid
  - Convert string or int to float: float()
  - Convert float or int to string: str()

- To check the data type of a variable: type()



How to round a floating point number to the desired number of decimal places?

We use round() function

```
x = 28793.54836
print('round(x)=',round(x)) #round to ones place
print('round(x, 2)=',round(x, 2)) # round to 2 decimal places
print('round(x, 1)=',round(x, 1)) #round to 1 decimal places
print('round(x, 0)=',round(x, 0)) #round to ones place, but return a float
print('round(x, -1)=',round(x, -1)) #round to tens place
print('round(x, -2)=',round(x, -2)) #round to hundreds place
#round(2.665,2), round(2.675,2)
```

executed in 6ms, finished 11:21:36 2020-10-10

```
round(x)= 28794
round(x, 2)= 28793.55
round(x, 1)= 28793.5
round(x, 0)= 28794.0
round(x, -1)= 28790.0
round(x, -2)= 28800.0
```

String Formatting

- Can be used to control the format of output strings
- It has many parameters, if you can't remember them all, make sure you understand the examples in the lectures and canvas->quiz exercises

```
:
print("You should {0} {1} what I say instead of what I {0}.".format("do", "only"))
print("The surname of {first} {last} is {last}.".format(first="John", last="Doe"))
```

executed in 46ms, finished 11:27:08 2020-10-10

```
You should do only what I say instead of what I do.
The surname of John Doe is Doe.
```

Operators

- Arithmetic operators
  - +, -, *, /, //, %
- Comparison operators
  - ==, >, <, !=
- Logical/boolean operators
  - and, or, not

= is different from ==
!= is different from not =

- Operator precedence and associativity
  - E.g., What is the return value of 4*5**2-5+4*2/2**2

- Compound Boolean expressions
  - E.g., what is the return value of 4<=5 and 5>=6
  - x <= y and x <= z is evaluated as (x <= y) and (x <= z)
  - What does x <= y <= z mean? It means x<=y and y<=z
  - What does x==y==z mean? It means x==y and y==z

int(x/y) is different from x//y:
When x and y are small integers, they may get the same result. But when they are big numbers, results may be different.

```
[3]: x=5
     y=1

     print(x//y)
     print(x/y)
     print(int(x/y))
```

```
5
5.0
5
```

```
[4]: x=25852016738884976640000
     y=1

     print(x//y)
     print(x/y)
     print(int(x/y))
```

```
25852016738884976640000
2.585201673888498e+22
25852016738884978212864
```

Why?
Again, floating point number is not stored precisely. When x is 25852016738884976640000, x/y returns a floating point number, but when the computer stores this floating point number, the value may be changed. So after we use int() to convert this floating point number, the value may be changed.

More references:
Reference 1
Reference 2

Equality

- To check whether two integers or strings equal or not, we use a==b
- To check whether two float number equal or not, we use math.isclose(a,b)

Truth table

When x and y are Boolean types

x	y	x and y	x or y	not x
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

When x and y are other data types

x	y	x and y	x or y	not x
not 0	doesn't matter	y	x	False
0	doesn't matter	x	y	True

short-circuit evaluation

Conditional execution

```
if condition :
    block
```

```
if condition :
    if-block
else:
    else-block
```

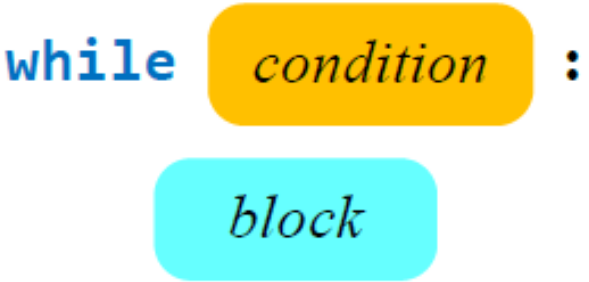
```
if condition-1 :
    block-1
elif condition-2 :
    block-2
elif condition-3 :
    block-3
elif condition-4 :
    block-4
    .
    .
    .
else:
    default-block
```

Tips

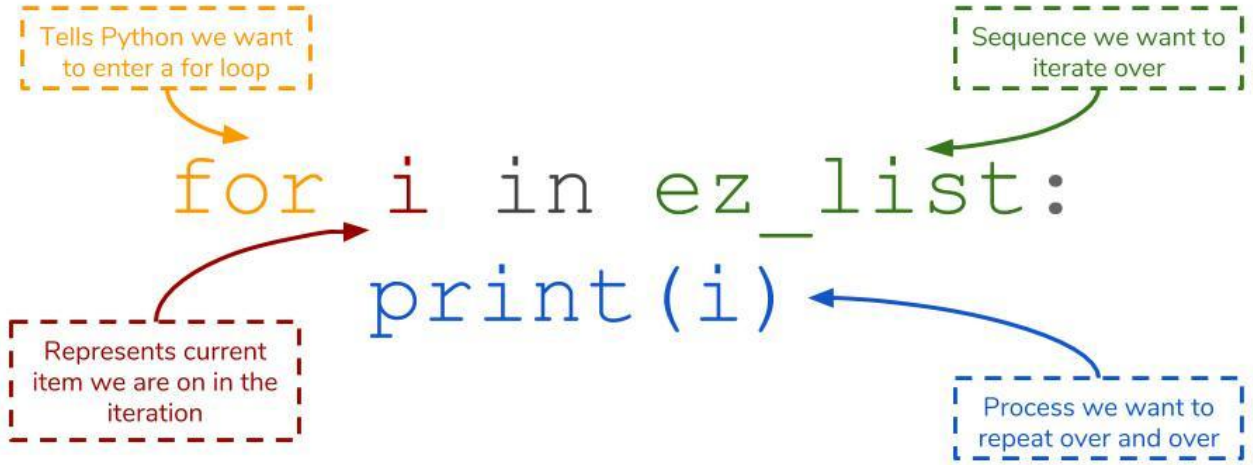
1. Don’t forget :
2. Don’t forget indentation

Iteration

while loop



for loop



Nested loop

- Useful to print something like a matrix

Listing 5.18: timestable3.py

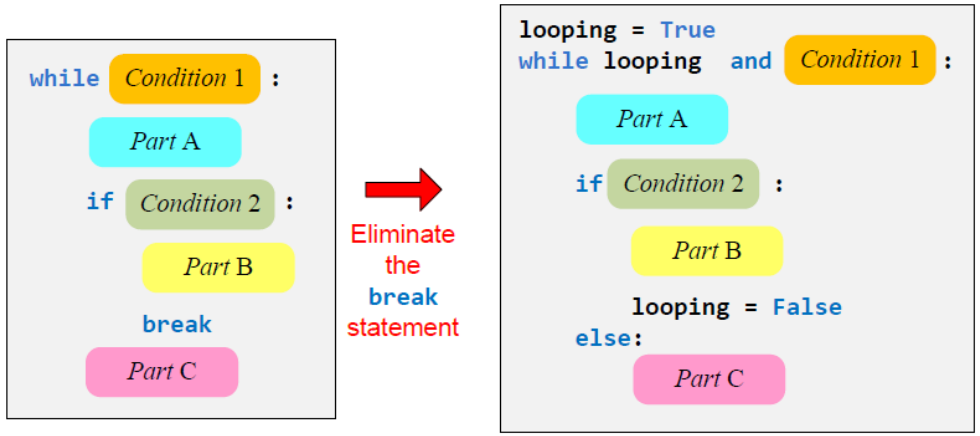
```
#  Get the number of rows and columns in the table
size = int(input("Please enter the table size: "))
#  Print a size x size multiplication table
for row in range(1, size + 1):
    for column in range(1, size + 1):
        product = row*column                    # Compute product
        print('{0:4}'.format(product), end='')  # Display product
    print()                                     # Move cursor to next row
```

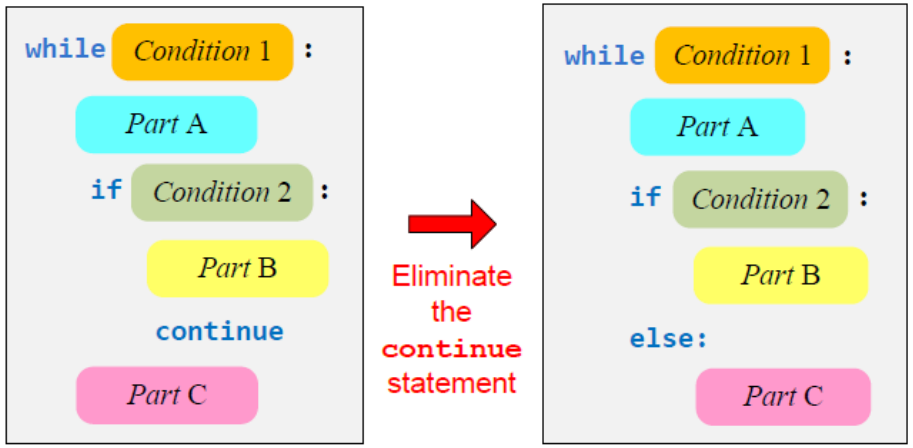
Listing 5.18 (timestable3.py) produces the table's contents in an attractive form:

```
Please enter the table size: 10
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
```

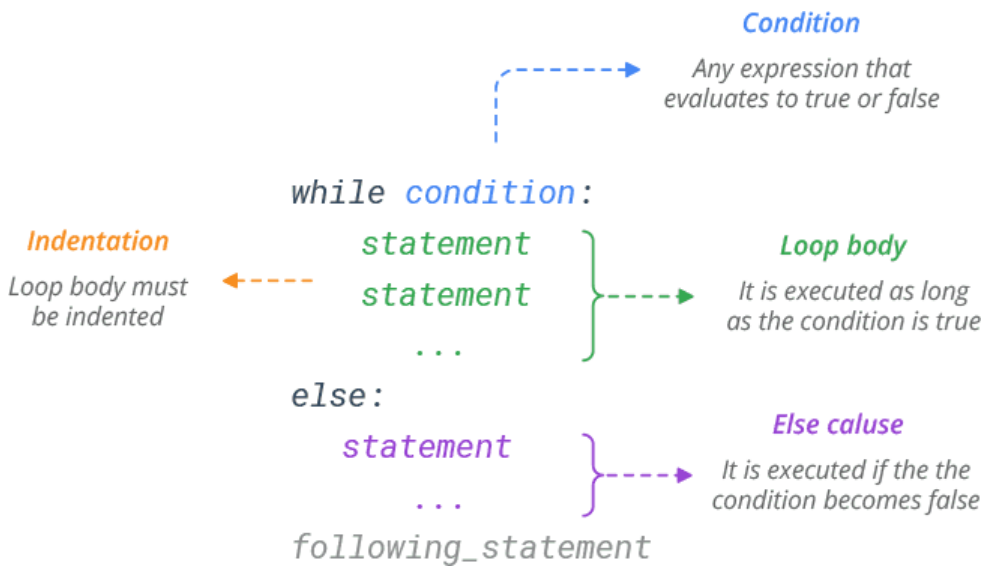
Break vs continue

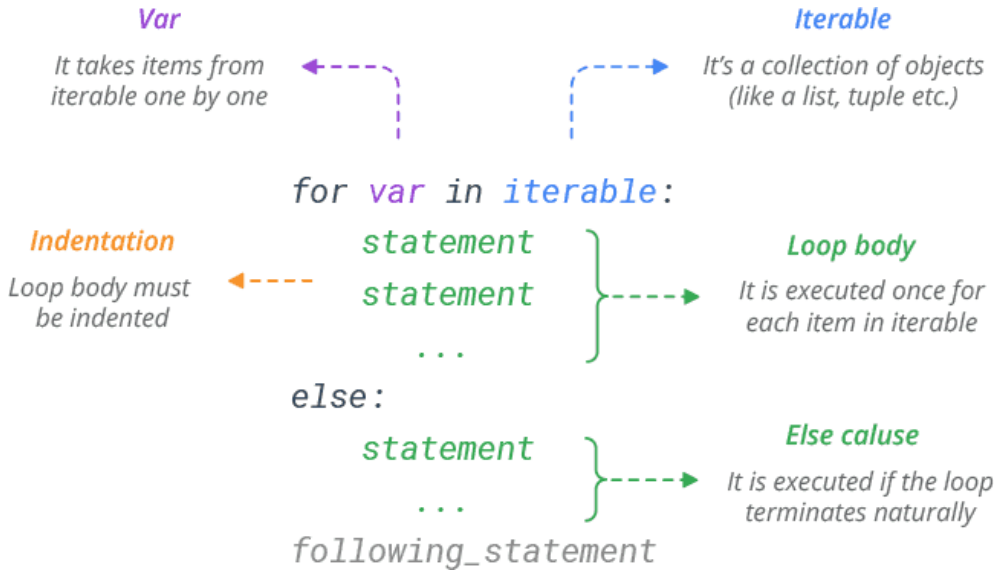
- break: terminate the whole loop
- continue: only terminate the current loop, go to next loop





while..else/for...else





Functions

Use functions

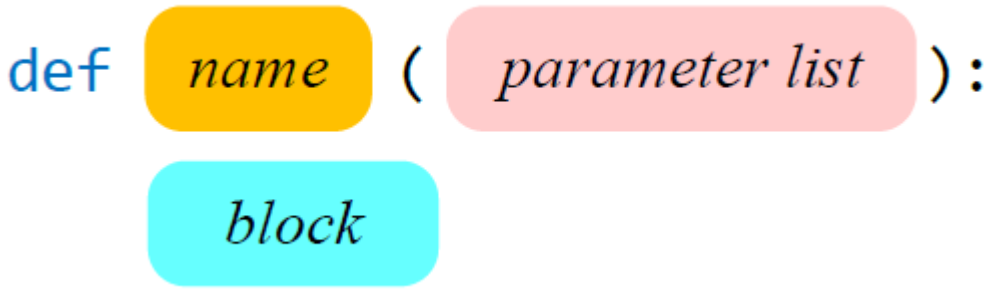
- To use functions defined by others, we need to import

- There’re different ways to import a function
  - from module_a import function_1
  - from module_a import *
  - from module_a import function_1 as f_1
  - import module_a.function_1 as f_1

Define functions

A function has three components:

- function name
- arguments or parameters (not a must)
- return data (not a must)



Many functions are introduced

- random()
- range()
- help()
- floor()
- ceil()
- log()
- …
- They’re introduced in different lectures, get familiar with them

Be familiar with some common operations

For example,

- How to decide if a number is even or odd?
- How to decide if a number is prime number using different methods?
- How to decide if a number is a factor of another number?
- How to decide if a number can be divisible by another number?
- ……

Objects

- Object-Oriented Programming
- Python is an object-oriented programming language
- Almost anything in python is an object

File objects

- how to create a directory and a file.
  - we use os.makedirs() function
- how to read data from a file.
  - we use open() function, and it has three modes. Be familiar with these modes
- how to write data to a file.
  - we use write() function.
- Remember to always close a file after you open it.
  - to eliminate this problem, we can use with statement because it will close the file automatically.
- how to delete a file.
  - we use os.remove() function

String objects

We introduce a lot of functions to manipulate strings, such as

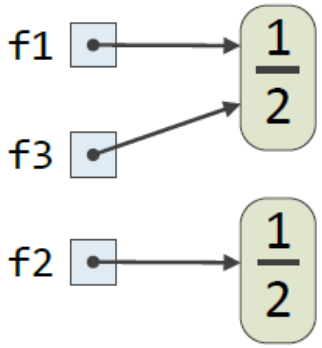
- How to search for a substring in a string?
  - find()
- How to split strings?
  - split()
- How to join strings?
  - join()
- How to remove both the leading and the trailing characters?
  - strip()
- How to convert upper-case strings to lower case strings, or vice versa?
  - upper()
  - lower()

Object aliasing

- what is object aliasing?
  - In Python, aliasing happens whenever one variable's value is assigned to another variable.

```
from fractions import Fraction

#  Assign some Fraction variables
f1 = Fraction(1, 2)
f2 = Fraction(1, 2)
f3 = f1
```



- how to verify if two objects x and y are identical?
  - x is y
  - id(x) == id(y)

Global variable vs Local variable vs nonlocal variable

- A local variable is a variable declared inside a function. It can be only accessed inside a function.
- A global variable is a variable declared outside of the function or in global scope. This means that a global variable can be accessed inside or outside of the function.
- A nonlocal variables is used in nested functions whose local scope is not defined. This means that the variable can be neither in the local nor the global scope. We use the nonlocal keyword to create nonlocal variables.

List/Tuples/Dictionary/Sets

- List is a collection which is ordered and changeable. Allows duplicate members.
- Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
- Dictionary is a collection which is unordered, changeable and indexed. No duplicate members.
- Set is a collection which is unordered and unindexed. No duplicate members.

You need to know

- how to create tuple/list/dictionary/set using different methods
  - Constructors, such as tuple() list() etc
  - Comprehensions
- how to access the elements
- Common operations about tuple/list/dictionary/set
  - such as delete, clear, sort, pop etc.
- Mutable vs immutable
  - Which data types are mutable and which data types are immutable

More on functions

- Recursion
- Optional arguments
- Variable number of arguments
- Generator
- Decorator

Recursion

Recursion is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem.

How to write a recursion function?

1) Find out all the base cases

2) Find out the recursion formula, i.e., how to calculate f(n) from previous calculations.

Fibonacci number: 0, 1, 1, 2, 3, 5, 8, 13, 21,....

Consider computing the Fibonacci number of order n:

$$F_n := \begin{cases} F_{n-1} + F_{n-2} & n > 1 \quad \text{(recurrence)} \\ 1 & n = 1 \quad \text{(base case)} \\ 0 & n = 0 \quad \text{(base case).} \end{cases}$$

Fibonacci numbers have practical applications in generating pseudorandom numbers.

Define a function `reverse_string()` that reverse a string `str`. Try using recursion.

For example:

Test	Result
`print(reverse_string('cityu'))`	`uytic`
`print(reverse_string('CS1302'))`	`2031SC`

Function arguments

Argument is a value passed to a function (or method) when calling the function. There are two types of arguments.

- keyword arguments
- positional arguments

There's another way to classify arguments:

- Required arguments
- Optional arguments

Variable number of arguments

def function_name(*args,**kwargs)

- *args (Non-Keyword Arguments)
  - args is a tuple of positional arguments.
- **kwargs (Keyword Arguments)
  - kwargs is a dictionary of keyword arguments.

Generator

- what's a generator
  - A generator is a programming object that produces (that is, generates) a sequence of values
- How to create a generator
  - use (iterable object)
  - use yield expression
- How to obtain the elements in generator?
  - use next()
  - use for loop
- How generator receive data from user?
  - use send()

Common mistake: if a question ask you to write a generator, you should use create a generator but not use print() to print the output on screen or return the output.

Decorator

- What's a decorator
  - A decorator is a function that takes another function and extends the behavior of the latter function without explicitly modifying it.How to create a generator

- How to define a decorator to decorate many functions?

Monte Carlo Simulation and Linear Algebra (not for exam)

- Monte Carlo simulation
  - What is Monte Carlo simulation
  - How to use it
- Linear Algebra
  - Some basic concepts in linear algebra
- Numpy
  - A very powerful Python library to solve different kinds of mathematical problems
  - Many functions are introduced
  - How to use Numpy to solve linear algebra problems

Important topics

- Conditional execution/if…else
- Functions (inner functions, global/nonlocal/local)
- Iteration/loop
- Recursion
- Generator
- Decorator
- Variable number of arguments
- List/tuple/set/dictionary

Midterm exam review

In thermodynamics, the Ideal Gas Law describes the behavior of a hypothetical ideal gas under various conditions. The pressure P (in atmospheres) exerted by the gas is given by the formula:

$$P = \frac{nRT}{V}$$

where:

- n is the amount of substance (in moles),
- T is the absolute temperature (in Kelvin),
- V is the volume of the container (in liters), and
- $R = 0.08206$ is the ideal gas constant.

Define a function `gas_pressure(n, T, V)` that takes the number of moles n, the temperature T, and the volume V all as `float`, and returns the expected pressure P as a `float`.

For example:

Test	Result
`print(f'{gas_pressure(1.0, 273.15, 22.414):.3f}')`	`1.000`
`print(f'{gas_pressure(2.5, 298.15, 10.0):.3f}')`	`6.117`
`print(f'{gas_pressure(0.5, 400.0, 50.0):.3f}')`	`0.328`

Answer: (penalty regime: 10, 20, ... %)

Reset answer

```
1  # YOUR CODE HERE
2  def gas_pressure(n, T, V):
3      return (n * 0.08206 * T) / V
```

According to general relativity, time passes slower the closer you get to a massive object like a black hole. If an observer far away from the black hole measures a time interval t_f, an observer sitting at a distance r from the center of a black hole of mass M will experience a time interval Δt given by:

$$\Delta t = t_f \sqrt{1 - \frac{2GM}{rc^2}}$$

where $G = 6.674 \times 10^{-11}$ is the gravitational constant, and $c = 3.0 \times 10^8$ is the speed of light.

Define a function `time_dilation(t_f, M, r)` that takes the far-away time t_f, the black hole mass M, and the distance r all as `float`, and returns the local time interval Δt as a `float`.

For example:

Test	Result
`print(f'{time_dilation(10.0, 1.0e30, 1.0e6):.3f}')`	`9.993`
`print(f'{time_dilation(1.0, 2.0e30, 3000.0):.3f}')`	`0.106`
`print(f'{time_dilation(50.0, 5.0e31, 1.0e6):.3f}')`	`48.110`

Answer: (penalty regime: 10, 20, ... %)

Reset answer

```
# YOUR CODE HERE
def time_dilation(t_f, M, r):
    G = 6.674e-11
    c = 3.0e8
    return t_f * (1 - (2 * G * M) / (r * (c ** 2)))**0.5
```

At standard atmospheric pressure, water changes its state of matter based on its temperature in degrees Celsius.

Write a function `water_state` that takes the temperature `T` of the water as a `float`. The function should return a `string` representing the water state based on the following rules:

- Return `'Solid'` if the temperature $T \leq 0$.
- Return `'Liquid'` if the temperature $0 < T < 100$.
- Return `'Gas'` if the temperature $T \geq 100$.

For example:

Test	Result
`print(water_state(2.5))`	`Liquid`
`print(water_state(-1.0))`	`Solid`
`print(water_state(102))`	`Gas`

Answer: (penalty regime: 10, 20, ... %)

Reset answer

```
# YOUR CODE HERE
def water_state(T):
    if T <= 0:
        return 'Solid'
    elif T>0 and T<100:
        return 'Liquid'
    else:
        return 'Gas'
```

Suppose in a city, the taxi fare for a ride is determined by the travel distance and the time period.

Write a function `taxi_fare` that takes

- a `float d` (the travel distance in meters), and
- a string `t` (the time period), where `t` is either `'day'` or `'night'`;

and returns the taxi fare as a `float`. The taxi fare is calculated as

$$\text{taxi fare} = d \times r,$$

where `r` is the rate given by

$$r = \begin{cases} 2.5 & t \text{ is 'day'}, \\ 3.0 & \text{otherwise.} \end{cases}$$

For example:

Test	Result
`print("{:.1f}".format(taxi_fare(1, 'day')))`	`2.5`
`print("{:.2f}".format(taxi_fare(3, 'night')))`	`9.00`

Answer: (penalty regime: 10, 20, ... %)

Reset answer

```
# YOUR CODE HERE
def taxi_fare(d, t):
    if t == 'day':
        r = 2.5
    else:
        r = 3.0
    return d * r
```

The exponential linear unit is an activation function in deep learning and is defined as

$$f(x) = \begin{cases} x & x \geq 0, \\ a(e^x - 1) & \text{otherwise.} \end{cases}$$

Write a function `elu(a)` that takes the parameter `a` as `float` and returns the exponential linear unit function that

- takes x as `float` and
- returns the value of $f(x)$ as `float`.

Hint: Import the `exp` function from the `math` module, where `exp(x)` returns e^x.

For example:

Test	Result

`g = elu(1.0)`
`print(f'{g(-1.0):.3f}')`
`print(f'{g(0.0):.3f}')`
`print(f'{g(1.0):.3f}')`
	`-0.632`
	`0.000`
	`1.000`

Answer: (penalty regime: 10, 20, ... %)

Reset answer

```
# YOUR CODE HERE
from math import exp
def elu(a):
    def f(x):
        if x>=0:
            return x
        else:
            return a*(exp(x)-1)
    return f
```

Write a program that continues to accept numbers from user input converted to `float` until the input is an empty string (i.e., the user presses `Enter` without typing anything). Then, the program should print the **third largest distinct number** (without additional formatting) among the inputs. If fewer than three distinct inputs are provided, then the program should print negative infinity `-inf`.

Hint: Negative infinity can be obtained using `float('-inf')`.

For example:

Input	Result
2 3.0 1.0 3.0 1 0	1.0

Answer: (penalty regime: 10, 20, ... %)

Reset answer

```
# YOUR CODE HERE
largest = second = third = float('-inf')
x=input()

while x:
    x = float(x)
    if x > largest:
        largest, second, third = x, largest, second
    elif largest > x > second:
        second, third = x, second
    elif second > x > third:
        third = x
    x=input()

print(third)
```

Write a boolean function `is_strong(n)` that takes a positive integer `n` and returns the boolean value `True` if and only if the sum of the factorials of its digits equals the number itself.

For example, `is_strong(145)` returns `True` since

$$145 = 1! + 4! + 5! = 1 + 24 + 120 = 145$$

Hint: You many use the function `factorial` defined in `math`.

For example:

Test	Result
`print(is_strong(145))`	`True`
`print(is_strong(146))`	`False`

Answer: (penalty regime: 10, 20, ... %)

Reset answer

```
# YOUR CODE HERE
from math import factorial

def is_strong(n):
    total = 0
    for digit in str(n):
        total += factorial(int(digit))
    return total == n

```

In many heat/diffusion models, the solution contains a Gaussian factor e^{-x^2}, which is often referred to as a heat kernel (Gaussian core).

The following is the Taylor series expansion of e^{-x^2} truncated up to the term with x^{2n} (i.e., the sum from $k = 0$ to $k = n$):

$$\sum_{k=0}^{n} \frac{(-1)^k x^{2k}}{k!} = 1 - \frac{x^2}{1!} + \frac{x^4}{2!} - \frac{x^6}{3!} + \cdots + (-1)^n \frac{x^{2n}}{n!}.$$

Define a function `heat_kernel(x, n)` that takes `x` as `float`, `n` as `int` ($n \geq 0$), and returns the value of the truncated series as float.

For example:

Test	Result
`print("{:.2f}".format(heat_kernel(1,1)))`	`0.00`
`print("{:.3f}".format(heat_kernel(0.5,2)))`	`0.781`

Answer: (penalty regime: 10, 20, ... %)

Reset answer

```
# YOUR CODE HERE
def heat_kernel(x, n):
    total = 1.0
    term = 1.0
    for k in range(1, n + 1):
        term *= (-x * x) / k
        total += term
    return total
```

- How to prepare for the final exam
  - Theory: review the basic knowledge by reviewing the lecture materials
  - Practice: Re-do all the programming questions in this course (exercises/labs/Equiz/mock exam).
- Please take some time to complete LOQ 
- You can try the mock exam.

Good Luck!