Introduction to

Sichen.Liu@xjtlu.edu.cn

XJTLU | SCHOOL OF FILM AND TV ARTS

Xi'an Jiaotong-Liverpool University 西交利物浦大学

Let us start

- If you like to follow along, you can open your own notebook. But please try to keep up with my presentation, as you still have time for exercises after the teaching.

Agenda

- Arithmetic operators
- Comparison operators
- Logical operators
- If Else
- Loops
- Exercises

Arithmetic operations

Arithmetic Operators and the **order of operation** are the same as in Mathematics.

We can also use round bracket ()

Symbol	Task Performed	Example	Result
+	Addition	4 + 3	7
-	Subtraction	4 - 3	1
*	Multiplication	4 * 3	12
**	Power of	7 ** 2	49
/	Division	7 / 2	3.5
//	Floor division	7 // 2	3
%	Mod	7 % 2	1

Order of operation example

```
16 ** 2 / 4
```

```
64.0
```

```
4 + 3 ** 2
```

13

vs

```
(4 + 3) ** 2
```

49

Comparison operators

- Return Boolean values

(i.e. True or False)

- Used extensively for conditional statements

Operator	Output
x == y	True if x and y have the same value
x != y	True if x and y don't have the same value
x < y	True if x is less than y
x > y	True if x is more than y
x <= y	True if x is less than or equal to y
x >= y	True if x is more than or equal to y

Comparison examples

```
x = 5        # assign 5 to the variable x
x == 5       # check if value of x is 5
```

```
True
```

Note that `==` *is not the same as* `=`

Logical operators

- Allows us to extend the conditional logic
- Will become essential later on

Operation	Result
x or y	True if at least on is True
x and y	True only if both are True
not x	True only if x is False

a	not a
False	True
True	False

a	b	a or b	a and b
False	False	False	False
False	True	True	False
True	False	True	False
True	True	True	True

Truth-table definitions of `bool` *operations*

Combining logical and comparison operators

```
x = 14
# check if x is within the range 10..20

( x > 10 ) and ( x < 20)
```

```
True
```

Another example

```
x = 14
y = 42

not (( x % 2 == 0 ) and ( y % 3 == 0))
```

```
False
```

That wasn't very easy to read was it?
Is there a way we can make it more readable?

Readable way

```
x = 14
y = 42

xDivisible = ( x % 2 ) == 0 # check if x is a multiple of 2
yDivisible = ( y % 3 ) == 0 # check if y is a multiple of 3

not (xDivisible and yDivisible)
```

```
False
```

If Else

- Fundamental build-in block of software

If Else example

Try running the example below.
What do you get?

```
Line 1  x = True
Line 2  if x:
Line 3      print("Executing if")
Line 4  else:
Line 5      print("Executing else")
Line 6  print("Prints regardless of the if-else block")
```

```
Executing if
Prints regardless of the if-else block
```

Indentation matters!

- Code is grouped by its indentation
- Indentation is the number of whitespace or tab characters before the code.
- If you put code in the wrong block, then you will get unexpected behavior

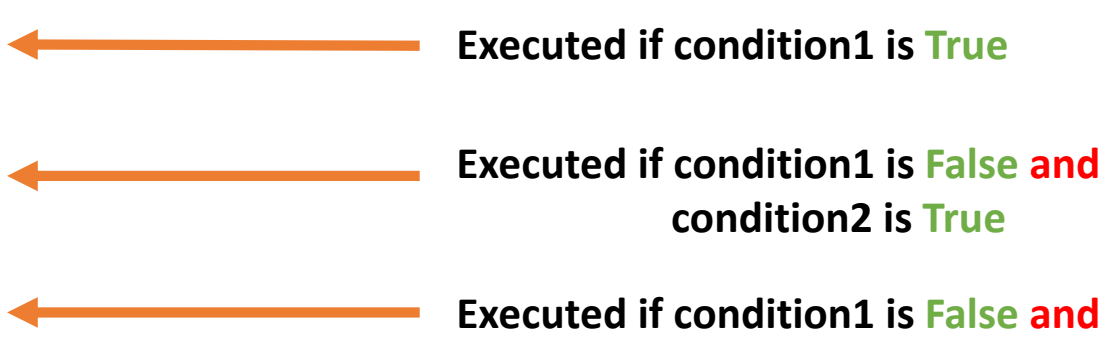
```
Line 1  x = True
Line 2  if x:
Line 3      print("Executing if")
Line 4  else:
Line 5      print("Executing else")
Line 6  print("Prints regardless of the if-else block")
```

```
Executing if
Prints regardless of the if-else block
```

Extending if-else blocks

- We can add infinitely more if statements using **elif**

```
if condition1:
    statement1
elif condition2:
    statement2
else:
    statement3
```



- elif = else + if which means that the previous condition must be false, then the current one will be check

Elif example

Try running the example below. What do you get?

```
age = 20

if age < 4:
    price = 0
elif age < 18:
    price = 5
elif age < 65:
    price = 10
else:
    price = 5

print("Your admission cost is " + str(price) + " RMB.")
```

```
Your admission cost is 10 RMB.
```

Omitting the else block

```
age = 20

if age < 4:
    price = 0
elif age < 18:
    price = 5
elif age < 65:
    price = 10
elif age >= 65:
    price = 5

print("Your admission cost is " + str(price) + " RMB.")
```

```
Your admission cost is 10 RMB.
```

For loop

- Allows us to iterate over numbers of variables within a data structure. During that we can manipulate each item

```
for item in itemList:
    do something to item
```

- Again, indentation is important here!

Example

- Say we want to go over a list and print each item along with its index

```
fruits = ["apple", "orange", "tomato", "banana"]
print("The fruit", fruits[0], "has index", fruits.index(fruits[0]))
print("The fruit", fruits[1], "has index", fruits.index(fruits[1]))
print("The fruit", fruits[2], "has index", fruits.index(fruits[2]))
print("The fruit", fruits[3], "has index", fruits.index(fruits[3]))
```

```
The fruit apple has index 0
The fruit orange has index 1
The fruit tomato has index 2
The fruit banana has index 3
```

- What if we have much more than 4 items in the list, say, 1000?

For example

- Now with a for loop

```
fruitList = ["apple", "orange", "tomato", "banana"]
for fruit in fruitList:
    print("The fruit", fruit, "has index", fruitList.index(fruit))
```

```
The fruit apple has index 0
The fruit orange has index 1
The fruit tomato has index 2
The fruit banana has index 3
```

- Saves us writing more lines
- Doesn't limit us in term of size

Numerical for loop

```
numbers = list(range(10))
for num in numbers:
    squared = num ** 2
    print(num, "squared is", squared)
```

```
0 squared is 0
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
6 squared is 36
7 squared is 49
8 squared is 64
9 squared is 81
```

While loop

- Another useful loop. Similar to the for loop.
- A while loop doesn't run for a predefined number of iterations, like a for loop. Instead, it stops as soon as a given condition becomes true/false.

```
n = 0
while n < 5:
    print("Executing while loop")
    n = n + 1

print("Finished while loop")
```

```
Executing while loop
Executing while loop
Executing while loop
Executing while loop
Executing while loop
Finished while loop
```

Break statement

- Allows us to go(break) out of a loop immediately.
- Adds a bit of controllability to a while loop.
- Usually used with an if.
- Can also be used in a for loop.

Quick quiz

How many times are we going to execute the while loop?

```
n = 0
while True:  # execute indefinitely
    print("Executing while loop")

    if n == 5:  # stop loop if n is 5
        break

    n = n + 1

print("Finished while loop")
```

Avoiding Infinite Loops

```
Executing while loop
Executing while loop
Executing while loop
Executing while loop
Executing while loop
Executing while loop
Finished while loop
```

Continue statement

```
num = 0
while num < 10:
    num = num + 1
    if num % 2 == 0:
        continue
    print(num)
```

```
1
3
5
7
9
```

Exercise time

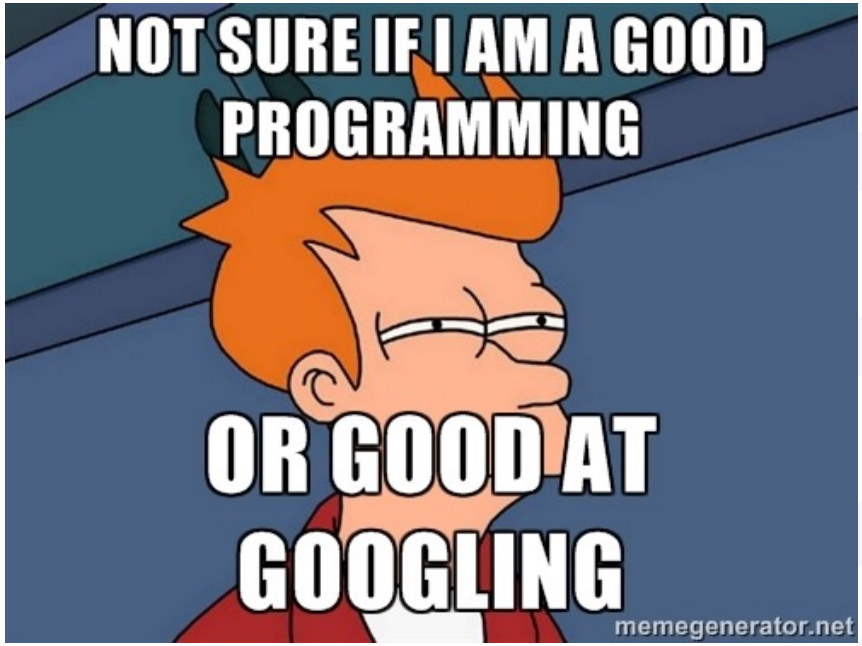
Simple and fun exercises.(Notebooks 3)

Failure is progress!

Ask us anything.

Ask among yourselves as well.

Google is your best friend when coding.



Let us start

- If you like to follow along, you can open your own notebook. But please try to keep up with my presentation, as you still have time for exercises after the teaching.

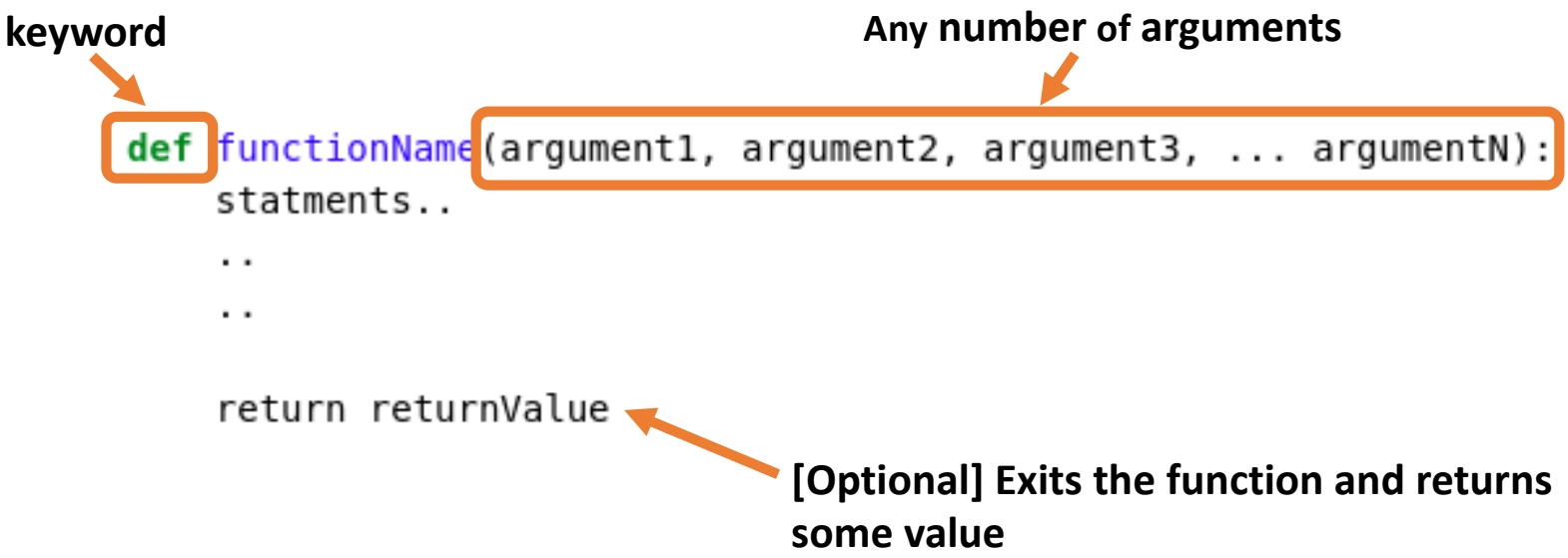
Agenda

- Functions
- Printing
- Classes
- Exercises

Functions

- Allow us to package functionality in a nice and readable way
- reuse it without writing it again
- Make code modular and readable
- Rule of thumb - if you are planning on using very similar code more than once, it may be worthwhile writing it as a reusable function.

Function declaration



- Functions accept arguments and execute a piece of code
- Often they also return values (the result of their code)

Without or with arguments

```
def greet_user():
    # Display a simple greeting.
    print("Hello!")

greet_user()
```

```
Hello!
```

Call the function

Add an argument

```
def greet_user(username):
    #Display a personalized greeting.
    print("Hello, " + username + "!")

greet_user('Laura')
greet_user('Lily')
greet_user('Mirra')
```

```
Hello, Laura!
Hello, Lily!
Hello, Mirra!
```

Default and Optional arguments

Default argument

```
def describe_pet(name, animal='dog'):
    # Display information about a pet.
    print("I have a " + animal + ".")
    print("Its name is " + name + ".")

describe_pet('Demon', 'cat')
describe_pet('Demon')
describe_pet(animal='cat', name='Demon')
```

```
I have a cat.
Its name is Demon.
I have a dog.
Its name is Demon.
I have a cat.
Its name is Demon.
```

Order doesn't matter

Optional argument

```
def describe_pet(name, animal=None):
    # Display information about a pet.
    if animal:
        print("I have a " + animal + ".")
    print("Its name is " + name + ".")

describe_pet('Demon', 'cat')
describe_pet('Demon')
```

```
I have a cat.
Its name is Demon.
Its name is Demon.
```

Function example

We want to make a function that rounds numbers up or down.

Try to pack the following into a function.

```
x = 3.4
remainder = x % 1
if remainder < 0.5:
    print("Number rounded down")
    x = x - remainder
else:
    print("Number rounded up")
    x = x + (1 - remainder)

print("Final answer is", x)
```

```
Number rounded down
Final answer is 3.0
```

Function example

```
def roundNum(num):
    remainder = num % 1
    if remainder < 0.5:
        return num - remainder
    else:
        return num + (1 - remainder)

# Will it work?
x = roundNum(3.4)
print (x)

y = roundNum(7.7)
print(y)

z = roundNum(9.2)
print(z)
```

```
3.0
8.0
9.0
```

Return multiple value

```
def listFunc(my_list):
    maximum = max(my_list)
    minimum = min(my_list)
    first = my_list[0]
    last = my_list[-1]
    return maximum, minimum, first, last

l = [24, 12, 68, 40, 120, 96]
params = listFunc(l)
print(params)
print("Max value is", params[0])
print("Min value is", params[1])
print("First value is", params[2])
print("Last value is", params[3])
```

```
(120, 12, 24, 96)
Max value is 120
Min value is 12
First value is 24
Last value is 96
```

Python built-in functions

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

To find out how they work: https://docs.python.org/3.7/library/functions.html

Printing

- When writing scripts, your outcomes aren't printed on the terminal.
- Thus, you must print them yourself with the print() function.
- Beware to not mix up the different type of variables!

```
print("Python is powerful!")
```

```
Python is powerful!
```

```
x = "Python is powerful"
y = " and versatile!"
print(x + y)
```

```
Python is powerful and versatile!
```

Quick quiz

Do you see anything wrong with this block?

```
str1 = "The string class has"
str2 = 76
str3 = "methods!"
print(str1 + str2 + str3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [25], in <cell line: 4>()
      2 str2 = 76
      3 str3 = "methods!"
----> 4 print(str1 + str2 + str3)

TypeError: can only concatenate str (not "int") to str
```

Another more generic way to fix it

```
print(argument1, argument2, argument3, .. ,argumentN)
```

```
str1 = "The string class has"
str2 = 76
str3 = "methods!"
print(str1, str2, str3)
```

```
The string class has 76 methods!
```

If we comma separate statements in a print function we can have different variables printing!

Placeholders

- A way to interleave numbers is

```
pi = 3.14159 # Pi
d = 12756 # Diameter of eath at equator (in km)
c = pi*d # Circumference of equator

#Print using +, and casting
print("Earth's diameter at equator: " + str(d) + " km. Equator's circumference: " + str(c) + " km.")
#Print using several arguments
print("Earth's diameter at equator:", d, "km. Equator's circumference: " , c, "km.")
#Print using .format alternative
print("Earth's diameter at equator: {:.1f} km. Equator's circumference: {:.1f} km.".format(d,c))
```

```
Earth's diameter at equator: 12756 km. Equator's circumference: 40074.12204 km.
Earth's diameter at equator: 12756 km. Equator's circumference:  40074.12204 km.
Earth's diameter at equator: 12756.0 km. Equator's circumference: 40074.1 km.
```

- Elegant and easy
- more in your notes

Commenting

- Useful when your code needs further explanation. Either for your future self and anybody else.
- Comments in Python are done with #
- Useful when you want to remove the code from execution but not permanently

- `print(totalCost)` is ambiguous and we can't exactly be sure what `totalCost` is.
- `print(totalCost)  # Prints the total cost for renovating the Main Library` is more informative

Classes

- Important for programming
- Useful, but more advanced
- Will not be taught here due to time limitations .. but there are explanations and examples in the notebooks

```
class className:
    globalValue = "global"
    # methods that belong to the class
    def __init__(self, name):
        # this method is called whenever a new instance is created
        self._instanceName = name

    def classMethod(self):
        # this is a method that belonds to the class
        # Note how we have an argument self, which is a reference to the object itself
```

Exercise time

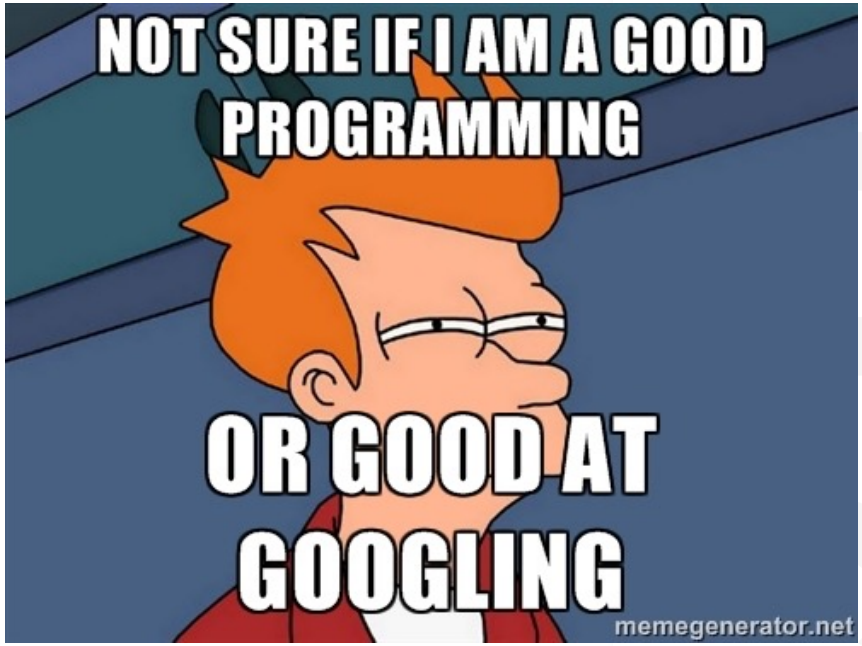
Simple and fun exercises.(Notebooks 4)

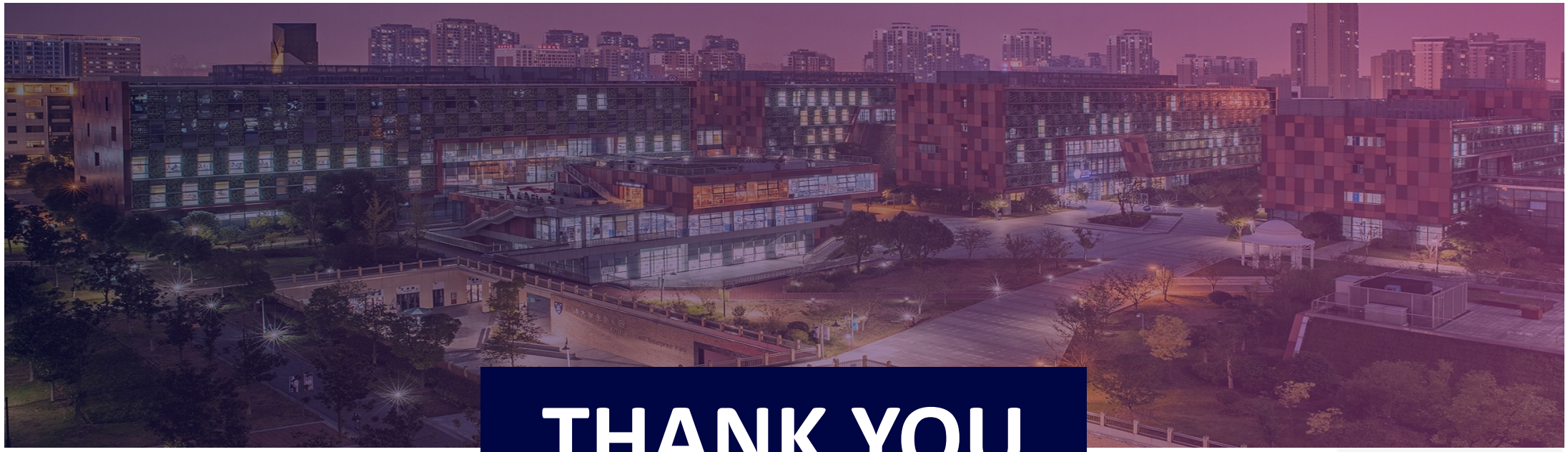
Failure is progress!

Ask us anything.

Ask among yourselves as well.

Google is your best friend when coding.





THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

@XJTLU

Xi'an Jiaotong-Liverpool University
西交利物浦大學

XJTLU | SCHOOL OF FILM AND TV ARTS