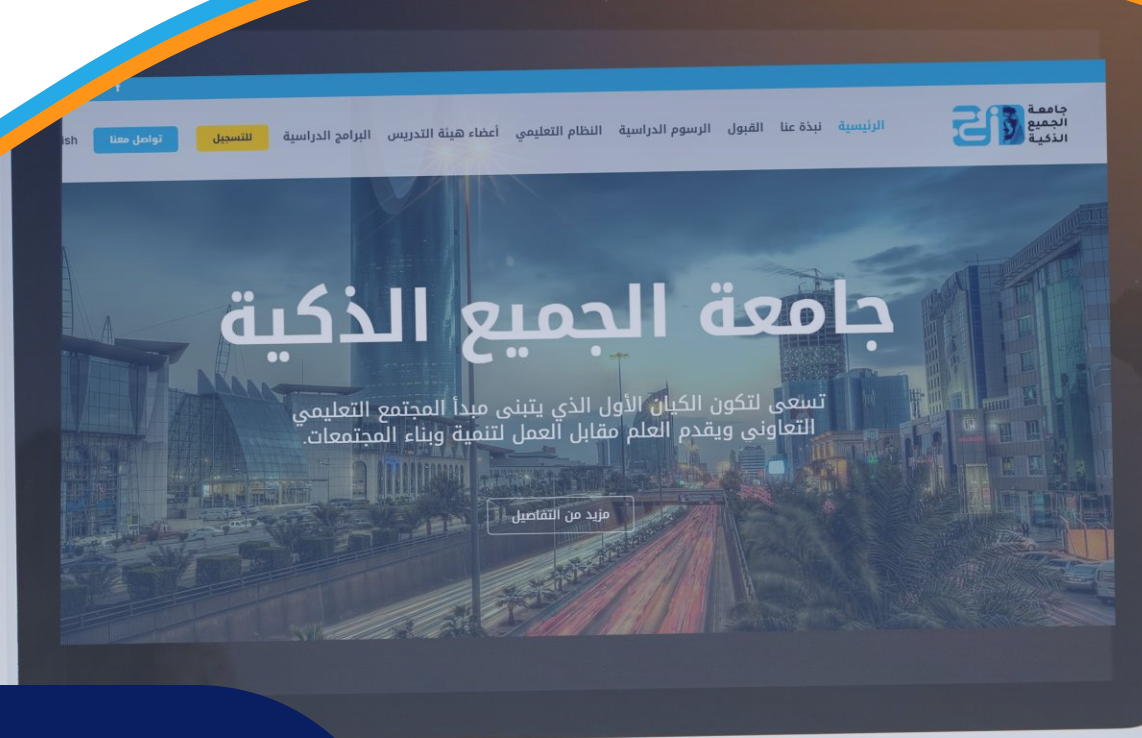




جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

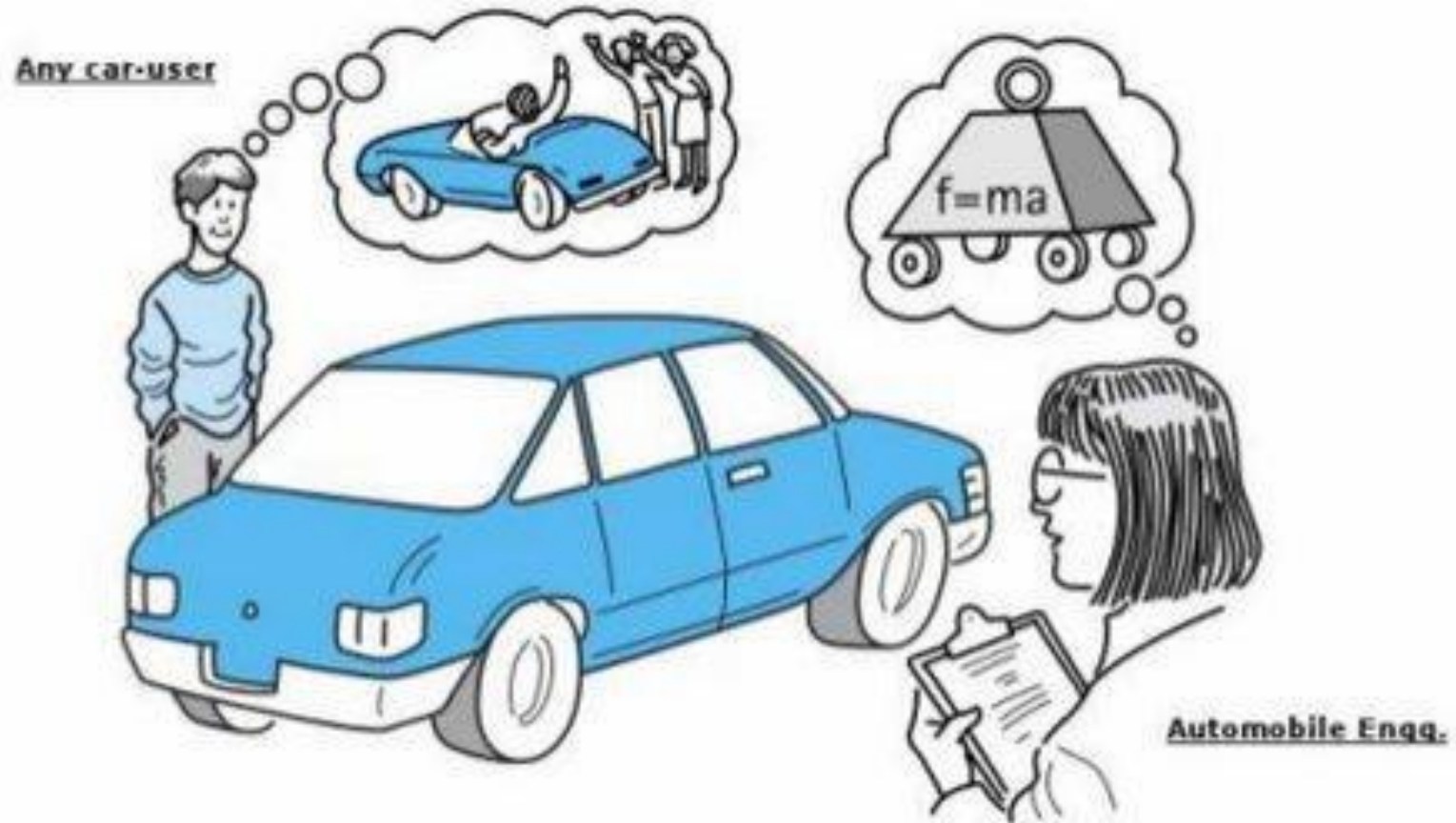


Functions in Python
Prepared by Dr. Hamada I. AbdulWakel
Lecture 6

CYB550
Programming Foundations In
Python

Lecture 6

Abstraction Mechanism



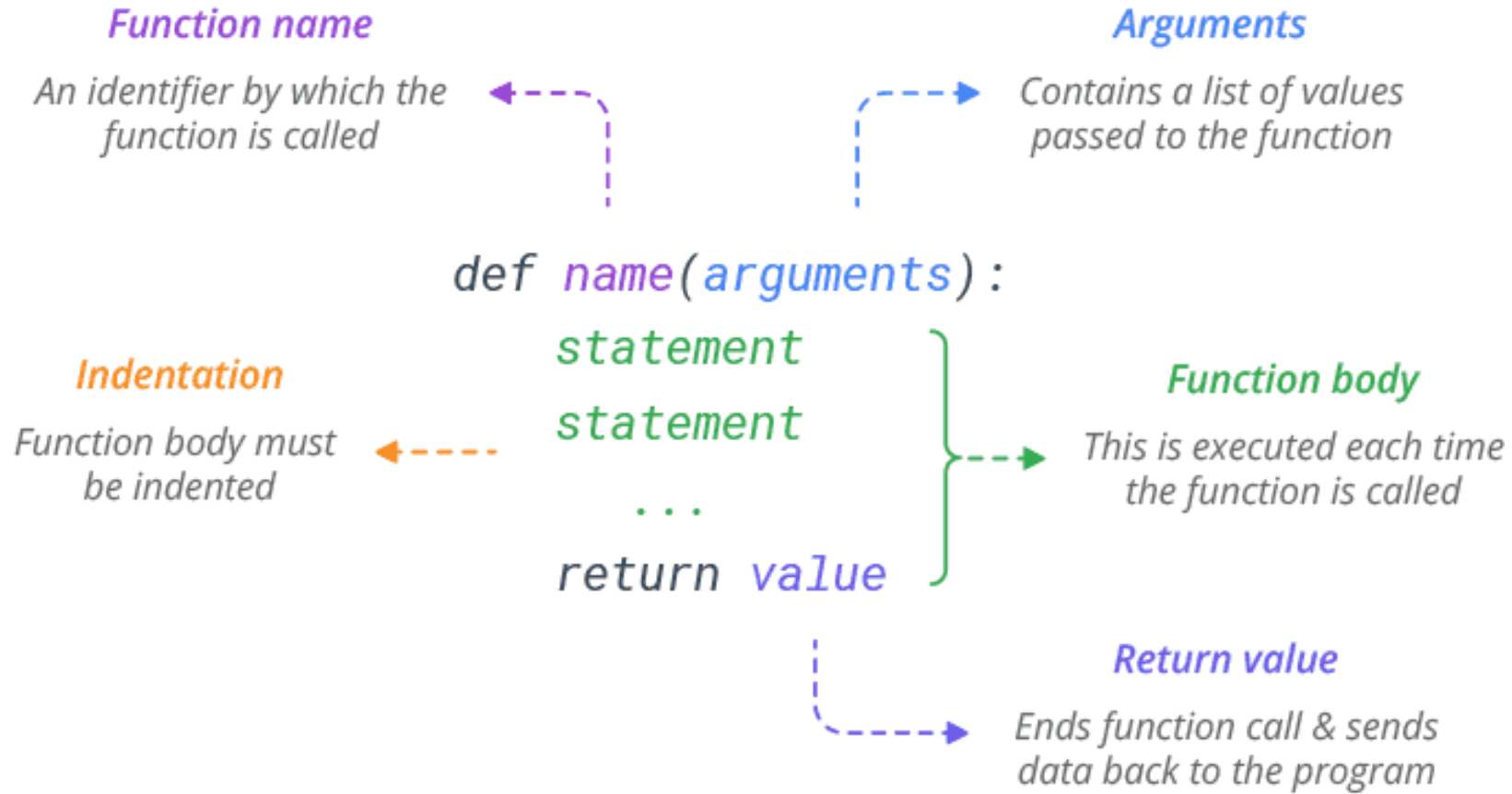
An abstraction includes the essential details relative to the perspective of the viewer

- Abstraction mechanism is used to hide details.
- Effective designers must invent useful abstractions to control complexity.

- Functions serve as abstraction mechanisms.
- Functions are the re-usable pieces of code which help us to organize structure of the code.
- Functions allow us to run a specific statements multiple times without repeating these statements.

Lecture 6

Creating Functions



- The def statement only creates a function but does not call it.
- After the def has run, you can call (run) the function by adding parentheses after the function's name.
 - Ex: `sum()`
`sum(start = s, end = e)`

- We can send information to a function by passing values, known as arguments/parameters.
- Arguments are declared after the function name in parentheses.

- A function may return a value via **the return statement**.
- Unlike C, C++ or Java, multiple values can be returned by a python function.
- If we don't include a return statement, the control will be transferred automatically to the calling code without returning any value. A default value of **None** will be returned if no return statement is there.
- To call a function we use **the function name** along with **arguments** that were used to define the function.



- Example1: Create function called greetings() that print a greeting message to the user.

```
def greetings():  
    print("Hello to my program :) ")
```

- Example2: Create function called sum() with two parameters start and end to calculate the sum of all the numbers starting from start to end.

```
def sum(start, end):  
    result = 0  
    for x in range(start, end+1):  
        result += x  
    print(result)
```

- Example2': Create function called sum() with two parameters start and end to calculate the sum of all the numbers starting from start to end.

```
def sum(start, end):  
    result = 0  
    If start > end:  
        print("End must be greater than start!!!!")  
        return # will return None  
    for x in range(start, end+1):  
        result += x  
    return result
```

- Example3:

```
def add(a, b):  
    return a + b
```

```
Res = add(5, 7) # will return 12
```

- Example4:

```
def hello():  
    print("Hello")
```

```
Res = hello() #will print Hello and return None
```

- Example5: returning multiple values

```
def bigger(a, b):  
    if a > b:  
        return a , b  
    else:  
        return b , a
```

```
Res = bigger(5, 7)  
print(Res , type(Res))
```

- Example6: write a function to get a class students marks and display the level of student (i.e., Above, Equal, or Below) the average.

```
students = [20, 25, 30, 65, 48, 35]
def stdAverage():
    sum = 0
    for std in students:
        sum += std
    avg = sum / len(students)
    for std in students:
        if std > avg:
            print("Above the average")
        elif std < avg:
            print("Below the average")
        else:
            print("Equal the average")
stdAverage()
```

- Example7: write a function to calculate and return the factorial $n!$ of a given number.

```
def fact(num):  
    prod = 1  
    for x in range(num, 1, -1):  
        prod *= x  
    return prod
```

- The local variable is the variable defined and used inside a code block.

```
def testLocal():  
    a = "Hello im a local variable :)"  
    print(a)  
  
print(a)
```

- The global variable is the variable defined and used inside the whole program.

```
a = "Hello im a global variable :)"  
def testGlobal():  
    print(a)
```

```
testGlobal()
```

- If you modify a global variable inside a function, this will throw an error called UnboundLocalError.

```
a = "Hello im a global variable :)"  
def testGlobal():  
    print(a)  # will throw an error  
    a = "Hello"  #this means a new variable  
    print(a)
```

testGlobal()

- If you need this modification, you need to explicitly tell python this.

```
a = "Hello im a global variable :)"
```

```
def testGlobal():
```

```
    global a
```

```
    print(a)
```

```
    a = "Hello"
```

```
    print(a)
```

```
testGlobal()
```

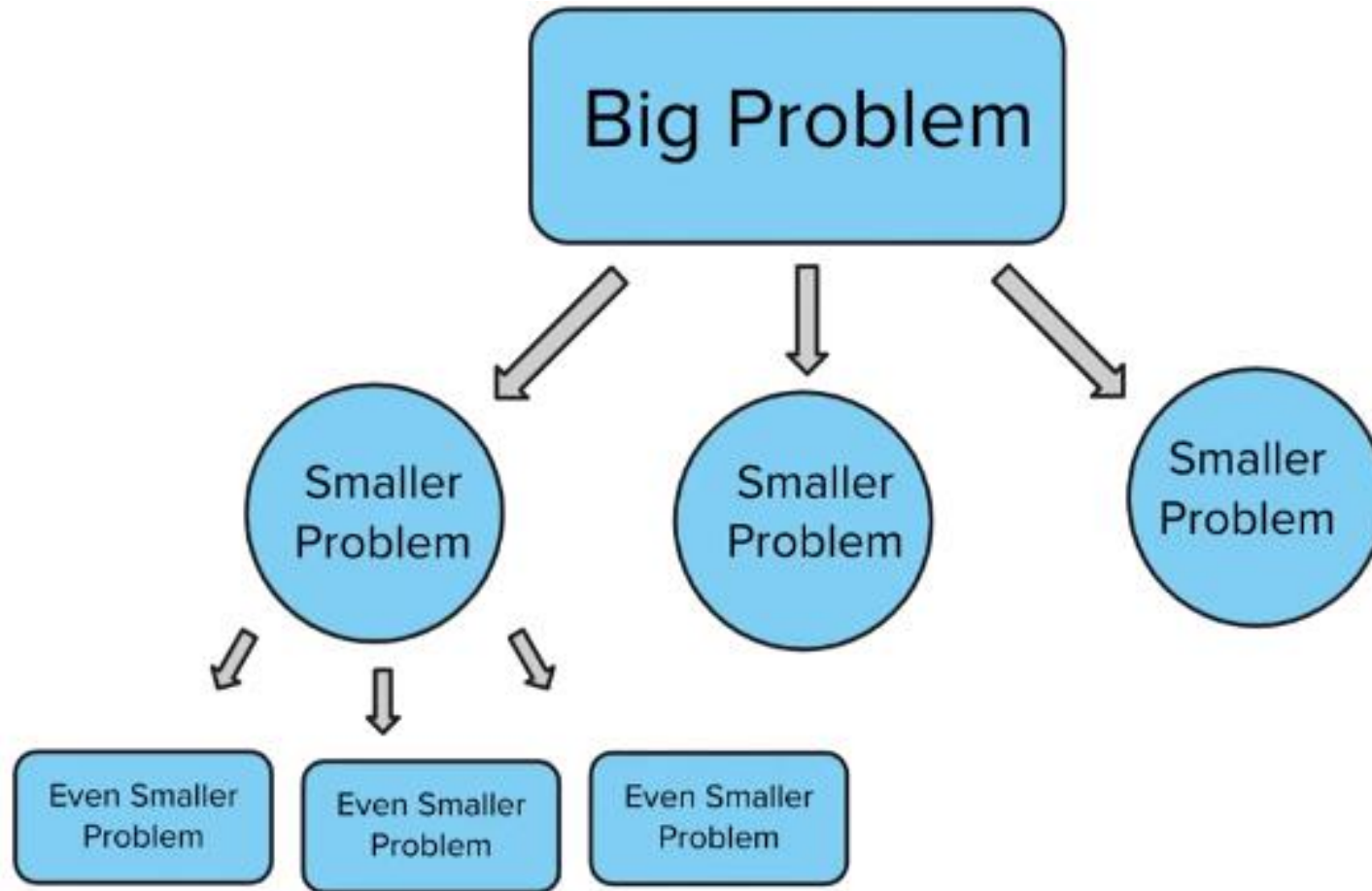


- We may quickly write a large amount of code, but later discover that large portions of the code are unnecessary. These issues emphasize the importance of **program design**.
- In a well-organized system, each part does its own job in collaborating to achieve a common goal.
- Each function should perform a single coherent task.

Example: Computing a summation.

Lecture 6

Problem Solving with Top-Down Design



- **Top-down design** (a.k.a stepwise refinement and problem decomposition) starts with a global view of the entire problem and breaks the problem into smaller, more manageable subproblems.
- The program that implements this design reflects the quality code's characteristics: readable, extendable, and debuggable.
- As functions are developed to solve subproblems, solution to overall problem is gradually filled out.

- Example8: Write a program to demonstrate the top-down code structure design.

def main():

```
    stdcount = int(input("Enter total number of students in your class: "))  
    studentsMarks = getMarks(stdcount)  
    markAverage = calAverage(studentsMarks)  
    displayStdLevel(studentsMarks, markAverage)
```

Lecture 6

Problem Solving with Top-Down Design

Cont'd

```
def getMarks(stds):  
    students = list()  
    for x in range(stds):  
        mark = int(input("Enter the student mark for student"+str(x+1)+": "))  
        students.append(mark)  
    return students
```

```
def calAverage(marks):  
    sum = 0  
    for x in marks:  
        sum += x  
    avg = sum / len(marks)  
    return avg
```



```
def displayStdLevel(Marks, average):  
    for x in Marks:  
        if x > average:  
            print("Above the average")  
        elif x < average:  
            print("Below the average")  
        else:  
            print("Equal the average")
```

```
main()
```

- Subproblems can be solved using the same function.
- Recursion can be used in place of looping by allowing a function to call itself.
- A recursive function has:
 - Base case: the smallest version of the problem.
 - Recursive case

- Example9: The factorial calculation:
 - Base case: 1 if $number = 0$
 - Recursive case: $number * (number - 1)!$ if $number > 0$

```
def fact(num):  
    if num == 0:  
        return 1  
    else:  
        return num * fact(num-1)
```

- Example10: a function to display the numbers from a lower bound to an upper one

```
def displayRange(lower, upper):  
    if lower <= upper:  
        print(lower)  
        displayRange(lower+1, upper)
```

- Example10: a function to display the Fibonacci sequence 1 1 2 3 5 8 13.....

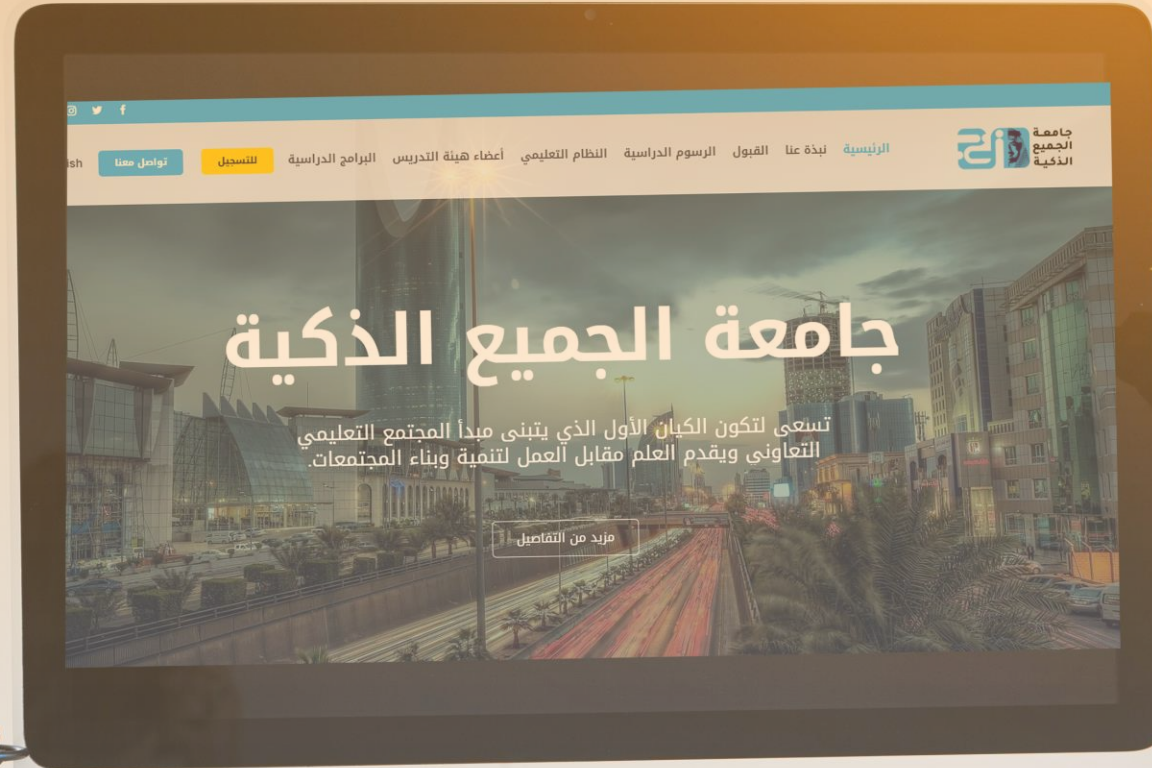
```
def calFib(number):  
    if number < 3:  
        return 1  
    else:  
        return calFib(number - 1) + calFib(number - 2)
```

- **Infinite recursion** arises when programmer fails to specify base case or to reduce size of problem in a way that terminates the recursive process.
 - Python virtual machine eventually runs out of memory resources to manage the process.

```
def runForever(number):  
    if number > 0:  
        runForever(number)  
    else:  
        return runForever(number - 1)
```



- Write a program using function to display the multiplication table of 1 to 5.
- Write a function to return the smallest number in a list. **Don't use build-in functions.**
- Write a program applying top-down approach to read the number of students and for each student the ID number and mark. Then displays the ID number and mark of the student with the highest mark.



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

THANK YOU