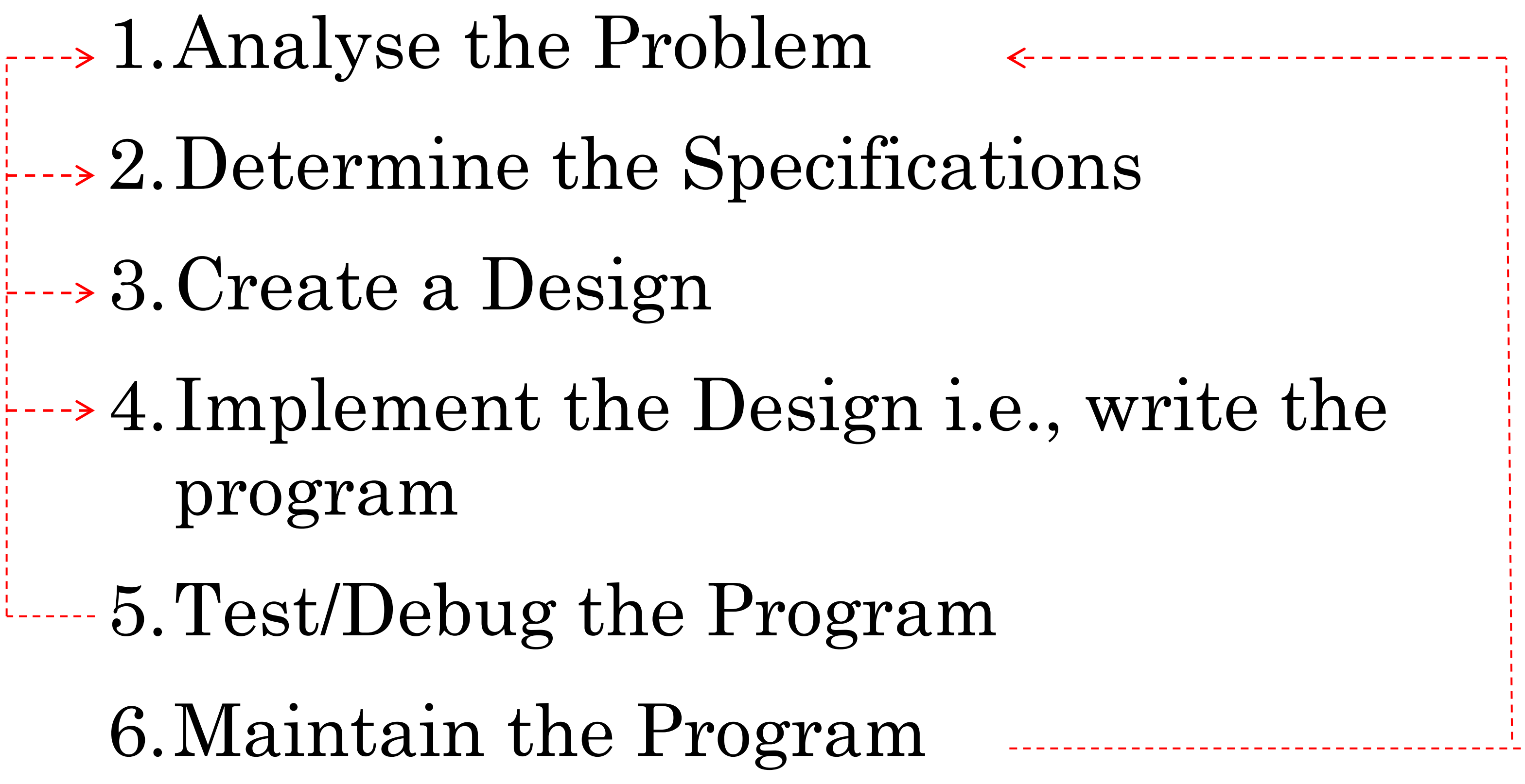
THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 6
Accumulator

Objectives of this Lecture

- A little revision
- To understand the accumulator program
- Example: Factorial

REVISION: The Software Development Process



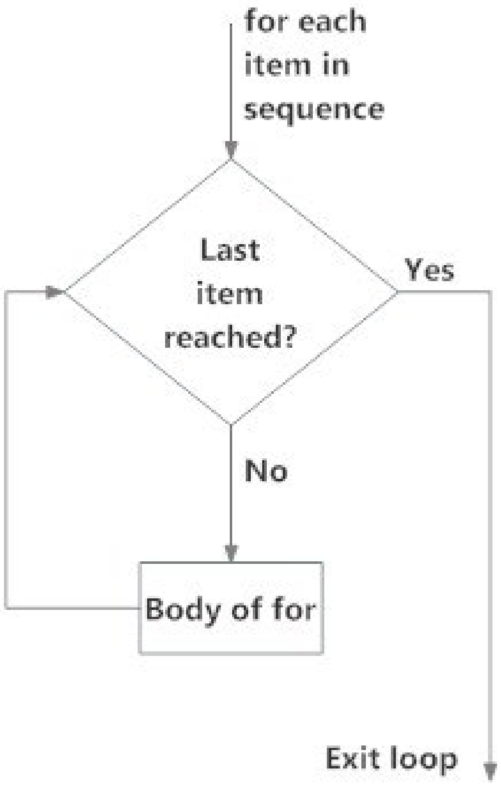
1. Analyse the Problem
2. Determine the Specifications
3. Create a Design
4. Implement the Design i.e., write the program
5. Test/Debug the Program
6. Maintain the Program

Software Life Cycle

REVISION: Definite Loops

for loops alter the flow of program execution, so they are referred to as control structures.

```
>>> for odd in [1, 3, 5, 7]:
          print(odd*odd)
1
9
25
49
```



Loop variable odd first has the value 1, then 3, then 5 and finally 7

Accumulator

- **Accumulator** is a variable used to accumulate (usually add up) values or perform calculations iteratively.
  - *keep track of changing values during a loop or sequence of operations*
- Pattern for Accumulators:
  - *Declare a variable of the appropriate type to store the running value*
  - *Initialize the accumulator to an appropriate value*
  - *Use a loop or iterative process to update the accumulator value as needed.*

Example Program: futval_gen.py

```
#    A program to compute the value of an investment
#    after specific number of years
#    Author: Unit Coordinator

def main():
    print("This program calculates the future")
    print("value for the investment after number of years.")

    principal = float(input("Enter the initial principal: "))
    apr = float(input("Enter the annual interest rate: "))
    yrs = int(input("Enter number of years: "))

    for i in range(yrs):
        principal *= (1 + apr)

    print ("The value in", yrs, "years is:", principal)

main()
```

Accumulation

Accumulating Results: Factorial

- Say you are waiting in a line with five other people. How many ways are there to arrange the six people?

- 720 -- which is the factorial of 6 (abbreviated 6!)

- Factorial is defined as:

  $n! = n(n-1)(n-2)\ldots(1)$

- So, 6! = 6*5*4*3*2*1 = 720

Accumulating Results: Factorial

- How could we write a program to do this?
- The basic outline of the program follows an input, process and output (IPO) pattern
  - *Input number to take factorial of, n*
  - *Compute factorial of n, fact*
  - *Output fact*

Accumulating Results: Factorial

- How did we calculate 6!?
- 6*5 = 30
- Take that 30, and 30 * 4 = 120
- Take that 120, and 120 * 3 = 360
- Take that 360, and 360 * 2 = 720
- Take that 720, and 720 * 1 = 720

Algorithm: Factorial

The general form of an accumulator algorithm looks like this:

- *Initialize the accumulator variable*
- *Perform computation*
  (e.g., in case of factorial multiply by the next smaller number)
- *Update accumulator variable*
- *Loop until final result is reached*
  (e.g., in case of factorial the next smaller number is 1)
- *Output accumulator variable*

Computational Thinking: Pattern recognition

- *pattern can be used repeatedly for range of problems*

Accumulating Results: Factorial

- It looks like we'll need a loop!

```
factorial = 1
for fact in [6, 5, 4, 3, 2, 1]:
   factorial = fact * factorial
```

- Let's trace through it to verify that this works!

Accumulating Results: Factorial

- Why did we need to initialize `factorial` to 1?
- There are a couple reasons…
  - *Each time through the loop, the previous value of* `factorial` *is used to calculate the next value of* `factorial`*. By doing the initialization, you know* `factorial` *will have a value the first time through.*
  - *If you use* `factorial` *without assigning it a value, what does Python do?*

Improving Factorial

- What does *`range(n)`* return?
  0, 1, 2, 3, …, n-1

- `range` has another optional parameter:
  - *`range(start, n)` returns start, start + 1, …, n-1*
  - *E.g., `range(1, 11)` returns: 1,2,3,4,5,6,7,8,9,10*

- But wait! There's more!

  `range(start, n, step)`
  returns: `start, start+step,` …stopping before `n`

- list(<sequence>) to make a list

Range()

- Let’s try some examples!

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(5,10))
[5, 6, 7, 8, 9]
>>> list(range(5,10,2))
[5, 7, 9]
```

range() Forwards or Backwards

- We can do the range for our loop a couple of different ways.
  - *We can count up from 2 to n:*
    `range(2, n+1)`
    (Why did we have to use n+1?)
  - *We can count down from n to 2:*
    `range(n, 1, -1)`

Back at the Factorial Program

Our completed factorial program:

```
#    Program to compute the factorial of a number
#    Illustrates for loop with an accumulator
#    Author: Unit coordinator

def factorial_find():
    n = int(input("Please enter an integer: "))
    factorial = 1
    for fact in range(n,1,-1):
       factorial = fact * factorial
    print("The factorial of", n, "is", factorial)
    return

factorial_find()
```

Lecture Summary

- We discussed an example of accumulator: factorial
- We discussed `range()` function