CSIT110 / CSIT810

Python

Lecture 12

Dr. Joseph Tonien

School of Computing and Information Technology
University of Wollongong

Objectives

Understanding of:

- Recursion

Recursion

A recursive function is a function that **calls itself.**

A recursive function usually has two steps:

- Base step: deals with **small cases**
- Recursion step: how a general case can be **derived from smaller cases**

Factorial function

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
```

Factorial function

1! = 1 → one factorial

2! = 1 x 2 = 2 → two factorial

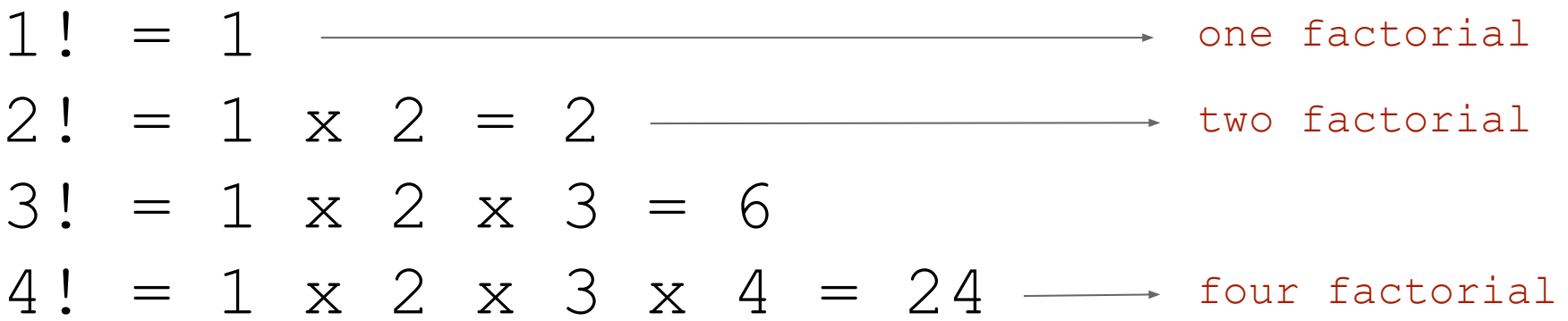
3! = 1 x 2 x 3 = 6

4! = 1 x 2 x 3 x 4 = 24 → four factorial

If we know 4! = 24,
how can we calculate 5! ?

5! = 4! x 5 = 24 x 5 = 120

Factorial function



In general, if we know factorial(n-1), we can calculate factorial(n) as:

factorial(n) = n x factorial(n-1)

Factorial function

```
# recursive factorial function
def factorial(n):
  if (n==1):
    return 1
  else:
    return n * factorial(n-1)
```

Factorial function

```
# recursive factorial function
def factorial(n):
    if (n==1):
        return 1
    else:
        return n * factorial(n-1)
```

base step

Factorial function

```
# recursive factorial function
def factorial(n):
  if (n==1):
    return 1
  else:
    return n * factorial(n-1)
```

recursion step

Factorial function

```
# recursive factorial function
def factorial(n):
  if (n==1):
    return 1
  else:
    return n * factorial(n-1)

for i in range(1,10):
  print("{0}! = {1}".format(i, factorial(i)))
```

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
```

Fibonacci sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

```
fibo(0) = 0
fibo(1) = 1
fibo(2) = 1
fibo(3) = 2
fibo(4) = 3
fibo(5) = 5
fibo(6) = 8
fibo(7) = 13
fibo(8) = 21
```

Fibonacci sequence

```
fibo(0) = 0
fibo(1) = 1
fibo(2) = 1
fibo(3) = 2
fibo(4) = 3
fibo(5) = 5
fibo(6) = 8
fibo(7) = 13
fibo(8) = 21
```

If we know
fibo(6) = 8 and fibo(7) = 13,
how can we calculate fibo(8) ?

fibo(8) = fibo(6) + fibo(7)
= 8 + 13 = 21

Fibonacci sequence

```
fibo(0) = 0
fibo(1) = 1
fibo(2) = 1
fibo(3) = 2
fibo(4) = 3
fibo(5) = 5
fibo(6) = 8
fibo(7) = 13
fibo(8) = 21
```

In general, we can calculate fibo(n) based on fibo(n-1) and fibo(n-2) as:

```
fibo(n) = fibo(n-1) + fibo(n-2)
```

Fibonacci sequence

```
# recursive function to calculate Fibonacci sequence
def fibo(n):
  if (n==0):
    return 0
  elif (n==1):
    return 1
  else:
    return fibo(n-1) + fibo(n-2)
```

Fibonacci sequence

```
# recursive function to calculate Fibonacci sequence
def fibo(n):
    if (n==0):
        return 0
    elif (n==1):
        return 1
    else:
        return fibo(n-1) + fibo(n-2)
```

Fibonacci sequence

```
# recursive function to calculate Fibonacci sequence
def fibo(n):
  if (n==0):
    return 0
  elif (n==1):
    return 1
  else:
    return fibo(n-1) + fibo(n-2)
```


recursion step

Fibonacci sequence

```
# recursive function to calculate Fibonacci sequence
def fibo(n):
  if (n==0):
    return 0
  elif (n==1):
    return 1
  else:
    return fibo(n-1) + fibo(n-2)

for i in range(0,10):
  print("fibo({0}) = {1}".format(i, fibo(i)))
```

```
fibo(0) = 0
fibo(1) = 1
fibo(2) = 1
fibo(3) = 2
fibo(4) = 3
fibo(5) = 5
fibo(6) = 8
fibo(7) = 13
fibo(8) = 21
fibo(9) = 34
```

Translate numerical code into words

“0278” → “zero-two-seven-eight”

“5” → “five”

“” → “”

“2000” → “two-zero-zero-zero”

Translate numerical code into words

Break the code into two parts

Part 1: the first character
Part 2: the rest of the string

Translate each part into words

And combine them

“2000”

“2” → “two”

“000” → “zero-zero-zero”

“two-zero-zero-zero”

Translate numerical code into words

We can only break the code into two parts
if it has at least two digits

base step: if the code is empty or has 1 digit

recursion step: if the code has at least 2 digits

Translate numerical code into words

base step: `if the code is empty or has 1 digit`

- `if the code is empty: easy`
- `if the code has 1 digit: ???`

Translate numerical code into words

- if the code has 1 digit:

```
NUMBER_MAPPING_DICT = {
  "0": "zero",
  "1": "one",
  "2": "two",
  "3": "three",
  "4": "four",
  "5": "five",
  "6": "six",
  "7": "seven",
  "8": "eight",
  "9": "nine"
}

# translate a digit into word using dictionary
def digit_to_word(digit):
  word = NUMBER_MAPPING_DICT[digit]
  return word
```

Translate numerical code into words

```
# translate numerical code into words
def numerical_to_word(numerical_string):

  if (len(numerical_string) == 0):
    # empty string

  elif (len(numerical_string) == 1):
    # only 1 digit

  else:
    # at least 2 digits
```

base step

recursion step

Translate numerical code into words

```
# translate numerical code into words
def numerical_to_word(numerical_string):

  if (len(numerical_string) == 0):
    # empty string

    return ""

  elif (len(numerical_string) == 1):
    # only 1 digit




  else:
    # at least 2 digits
```

Translate numerical code into words

```
# translate numerical code into words
def numerical_to_word(numerical_string):

  if (len(numerical_string) == 0):
    # empty string

    return ""

  elif (len(numerical_string) == 1):
    # only 1 digit
    word = digit_to_word(numerical_string)

    return NUMBER_MAPPING_DICT[numerical_string]

  else:
    # at least 2 digits
```

Translate numerical code into words

```
else:
    # at least 2 digits

    # break the string into two parts:

    # the first character
    part1 = numerical_string[0]

    # substring from the second character
    part2 = numerical_string[1:]

    # translate the two parts into words
    part1_word = numerical_to_word(part1)
    part2_word = numerical_to_word(part2)

    # combine them
    word = part1_word + "-" + part2_word

    return word
```


recursion step

Translate numerical code into words

```
# main program

# ask user for a numerical string
user_input = input("Enter a numerical string: ")

# translate into english words
word = numerical_to_word(user_input)

# display it
print(word)
```