CSIT881
Programming and Data Structures

Function

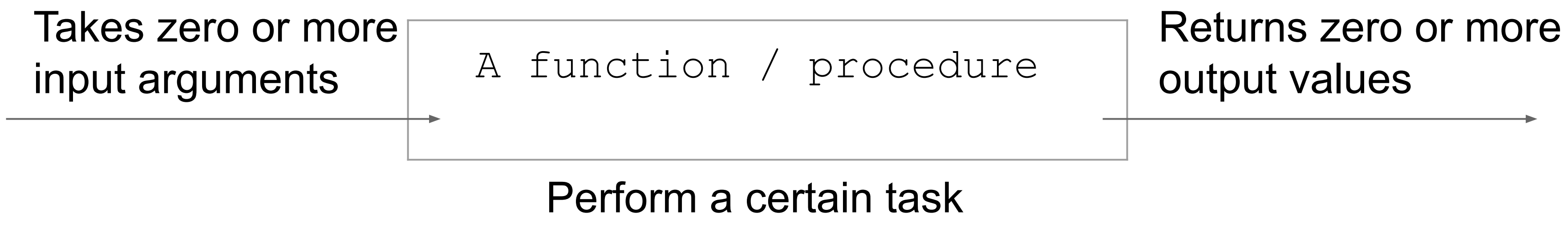
UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Dr. Joseph Tonien

Objectives

- Write your own functions
- Recursive functions
- Useful functions:
  - Rounding off function
  - Max and min functions
  - Random function

Function


Takes zero or more input arguments
A function / procedure
Returns zero or more output values
Perform a certain task

Function declaration

```
def function_name(arg1, arg2, arg3, …, argN):

  … perform a certain task …

  return value1, value2, value3, …, valueM
```

Function

Function declaration

```
def function_name(arg1, arg2, …, argN):

  … perform a certain task …

  return value1, value2, …, valueM
```

Calling function:

- providing correct number of arguments;
- using correct number of variables to save the return values

```
var1, var2, …, varM = function_name(arg1, arg2, …, argN)
```

Function

Calling the function examples:

function has 2 arguments and returns 1 value

```
variable_name = function_name(arg1, arg2)
```

function has 2 arguments and returns 2 values

```
var1, var2 = function_name(arg1, arg2)
```

function has 3 arguments and returns 0 values

```
function_name(arg1, arg2, arg3)
```

function has 0 arguments and returns 1 value

```
variable_name = function_name()
```

function has 0 arguments and returns 0 values

```
function_name()
```

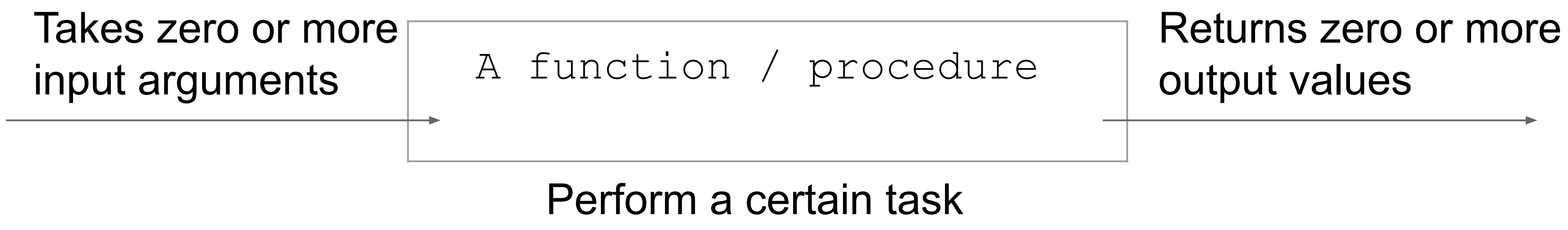
Terminology

Same meaning:
- Function
- Procedure
- Method
- Routine

Same meaning:
- Parameters
- Arguments
- Input values
- Input

Same meaning:
- Return values
- Output values
- Output

Function


Takes zero or more input arguments
A function / procedure
Returns zero or more output values
Perform a certain task

When we design a function, we need to ask the following questions:

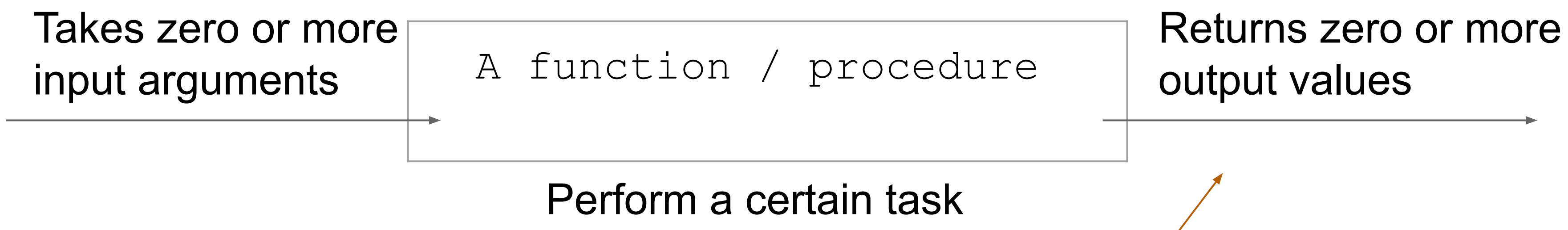
- What information does the function need to know in order to do its job?

  This will determine how many input arguments the function takes

  For example, if the job of a function is to add two numbers, then this function needs to know the two numbers. So the function will have 2 input arguments.

```
# calculate sum of two numbers
def add_two_numbers(number1, number2):
    ...
```

Function


Takes zero or more input arguments
A function / procedure
Returns zero or more output values
Perform a certain task

When we design a function, we need to ask the following questions:

- What information does the function give back?

  This will determine how many return values

  For example, if the job of a function is to add two numbers, then this function will give back the sum. So the function will return 1 value.

```
# calculate sum of two numbers
def add_two_numbers(number1, number2):
  number_sum = number1 + number2
  return number_sum
```

Function

```python
# calculate sum of two numbers
def add_two_numbers(number1, number2):
  number_sum = number1 + number2
  return number_sum
```

```python
# calling function
number1 = add_two_numbers(4, 3)               → 7

number2 = add_two_numbers(2, number1)         → 9

number3 = add_two_numbers(number2, number2)   → 18

number4 = add_two_numbers(number3, 10)        → 28

print(number4)
```

Mark and grade

At a fictional college, the following grading scheme is used:

Mark	Grade
100 - 80	A
79 - 60	B
59 - 40	C
39 - 0	D

```
Please enter mark: 90
Mark 90, Grade A
```

```
Please enter mark: 62
Mark 62, Grade B
```

```
Please enter mark: 5
Mark 5, Grade D
```

Mark and grade

```
# calculate grade based on mark
def calculate_grade(mark):
  grade = "frog"
  return grade
```

```
# ask user to enter mark
mark_input = input("Please enter mark: ")
mark = int(mark_input)

# determine grade based on mark
grade = calculate_grade(mark)

# display mark and grade
print("Mark {0}, Grade {1}".format(mark, grade))
```

```
Please enter mark: 90
Mark 90, Grade frog
```

Mark and grade

```python
# calculate grade based on mark
def calculate_grade(mark):
  grade = "frog"
  return grade
```

rewrite

```python
def calculate_grade(mark):
  #grade A: 100-80, B: 79-60, C: 59-40, D: 39-0

  if (mark >= 80):
    grade = "A"
  elif (mark >= 60):
    grade = "B"
  elif (mark >= 40):
    grade = "C"
  else:
    grade = "D"

  return grade
```

```
Please enter mark: 90
Mark 90, Grade A
```

Mark and grade

```
def calculate_grade(mark):
  if (mark >= 80):
    grade = "A"
  elif (mark >= 60):
    grade = "B"
  elif (mark >= 40):
    grade = "C"
  else:
    grade = "D"

  return grade
```

this is the same

```
def calculate_grade(mark):
  if (mark >= 80):
    return "A"
  elif (mark >= 60):
    return "B"
  elif (mark >= 40):
    return "C"

  return "D"
```

Mark and grade

```
def calculate_grade(mark):
  ...

  return grade
```

- How many input arguments/parameters does this function take? And why?
  - This function takes 1 input argument / parameter.
  - Reason: in order to determine the grade, the function needs to know the mark.
- How many output values does this function return?
  - This function returns 1 value (which is the grade).

Hello!

```
Enter first name: John
Enter last name: Smith
Hello John Smith!
```

```
# ask user for name
first_name, last_name = ask_name()

# display greeting
say_hello(first_name, last_name)
```

Hello!

```python
# ask user for name
def ask_name():
  first_name = "Finley"
  last_name = "Fish"
  return first_name, last_name
```

```python
# display greeting
def say_hello(first_name, last_name):
  print("Hello {0} {1}!".format(first_name, last_name))
```

```python
# ask user for name
first_name, last_name = ask_name()

# display greeting
say_hello(first_name, last_name)
```

```
Hello Finley Fish!
```

Hello!

```
# ask user for name
def ask_name():
  first_name = input("Enter first name: ")
  last_name = input("Enter last name: ")

  return first_name, last_name
```

```
# display greeting
def say_hello(first_name, last_name):
  print("Hello {0} {1}!".format(first_name, last_name))
```

```
# ask user for name
first_name, last_name = ask_name()

# display greeting
say_hello(first_name, last_name)
```

```
Enter first name: John
Enter last name: Smith
Hello John Smith!
```

Hello!

```
# ask user for name
def ask_name():
  ...

  return first_name, last_name
```

- How many input arguments/parameters does this function take? And why?
  - This function takes 0 input arguments / parameters.
  - Reason: the function does not need to know anything to perform its task!

- How many output values does this function return?
  - This function returns 2 values (which are the first and last name).

Hello!

```
# ask user for name
def ask_name():
  ...

  return first_name, last_name
```

```
# ask user for name
first_name, last_name = ask_name()

# display greeting
say_hello(first_name, last_name)
```

- Why do we have to write
  `first_name, last_name = ask_name()` ?
  - Reason: the function returns 2 values, so we need to save them into 2 variables `first_name` and `last_name`

Hello!

```
# display greeting
def say_hello(first_name, last_name):
  print("Hello {0} {1}!".format(first_name, last_name))
```

- How many input arguments/parameters does this function take? And why?
  - This function takes 2 input arguments / parameters.
  - Reason: the function needs to know both first name and last name to display the greeting message.

- How many output values does this function return?
  - This function returns 0 values. That is why we do not need to use the **`return`** statement.

Expanding word

```
Enter a word: Meow
Enter expand factor: 4
Here you go: MMMMeeeeoooowwww
```

```
Enter a word: Cat
Enter expand factor: 2
Here you go: CCaatt
```

```
Enter a word: Dog
Enter expand factor: 1
Here you go: Dog
```

```
Enter a word: Frog
Enter expand factor: 0
Here you go:
```

Expanding word

```
Enter a word: Cat
Enter expand factor: 2
Here you go: CCaatt
```

```
Initially set expand_word = ""

original letter   expand
     C                CC     expand_word = "CC"
     a                aa     expand_word = "CCaa"
     t                tt     expand_word = "CCaatt"
```

Expanding word

```
Enter a word: Cat
Enter expand factor: 2
Here you go: CCaatt
```

```
# ask user for input
word, multiplicity = ask_input()

# expand the word
new_word = expand(word, multiplicity)

# display the result
print("Here you go: " + new_word)
```

Expanding word

```
# ask user for input
def ask_input():
  word = "frog"
  multiplicity = 5
  return word, multiplicity
```

```
# expand the word
def expand(word, multiplicity):
  result = "Hey Google - How much wood would a woodchuck chuck if a woodchuck could chuck wood?"
  return result
```

```
# ask user for input
word, multiplicity = ask_input()

# expand the word
new_word = expand(word, multiplicity)

# display the result
print("Here you go: " + new_word)
```

Expanding word

```python
# ask user for input
def ask_input():
  word = "frog"
  multiplicity = 5
  return word, multiplicity
```

rewrite

```python
def ask_input():
  # ask a word
  word = input("Enter a word: ")

  # ask expand factor
  user_input = input("Enter expand factor: ")
  multiplicity = int(user_input)

  return word, multiplicity
```

Expanding word

```
# expand the word
def expand(word, multiplicity):
  result = "Hey Google - How much wood would a woodchuck chuck if a woodchuck could chuck wood?"
  return result
```

rewrite

```
def expand(word, multiplicity):
  # initialize result as empty string
  result = ""

  for i in range(0, len(word)):
    # get the ith letter from the word
    letter = word[i]

    # multiply the letter
    letter_multiply = letter * multiplicity

    # adding the expanded letter to the result
    result = result + letter_multiply

  return result
```

Generate password

In an online game, the initial password is generated from the username by replacing each letter i to 1, r to 7, s to 5, and z to 2.

Write a program to generate this initial password.

```
Enter username: Superman123
Password is 5upe7man123
```

```
Enter username: zebra8
Password is 2eb7a8
```

```
Initially set password = ""
Username letter   Password letter
     z                 2               password = "2"
     e                 e               password = "2e"
     b                 b               password = "2eb"
     r                 7               password = "2eb7"
     a                 a               password = "2eb7a"
     8                 8               password = "2eb7a8"
```

Generate password

```
# construct the password for username
def generate_password(username):
  password = "frog"
  return password
```

```
# ask user to enter username
username = input("Enter username: ")

# construct the password
password = generate_password(username)

# display password result
print("Password is " + password)
```

```
Enter username: zebra8
Password is frog
```

Generate password

```
# construct the password for username
def generate_password(username):
  password = "frog"
  return password
```

rewrite

```
def generate_password(username):
  # initialize password as empty string
  password = ""

  for i in range(0, len(username)):
    # get the ith character from username
    username_letter = username[i]

    # construct corresponding character for password
    password_letter = transform_character(username_letter)

    # adding a character to password
    password = password + password_letter

  return password
```

Generate password

```
# construct password letter from username letter
def transform_character(letter):
  password_letter = "p"
  return password_letter
```

```
Enter username: zebra8
Password is pppppp
```

```
def generate_password(username):
  # initialize password as empty string
  password = ""

  for i in range(0, len(username)):
    # get the ith character from username
    username_letter = username[i]

    # construct corresponding character for password
    password_letter = transform_character(username_letter)

    # adding a character to password
    password = password + password_letter

  return password
```

Generate password

```
# construct password letter from username letter
def transform_character(letter):
  password_letter = "p"
  return password_letter
```

rewrite

```
def transform_character(letter):
  if (letter == "i") or (letter == "I"):
    password_letter = "1"
  elif (letter == "r") or (letter == "R"):
    password_letter = "7"
  elif (letter == "s") or (letter == "S"):
    password_letter = "5"
  elif (letter == "z") or (letter == "Z"):
    password_letter = "2"
  else:
    password_letter = letter

  return password_letter
```

```
Enter username: zebra8
Password is 2eb7a8
```

Default arguments

Function arguments can have default values. If the function is called without an argument, the argument gets its default value.

```
# display a welcome message
def welcome(name, greeting="Hi"):
#{
  print("{0} {1}!".format(greeting, name))
#}
```

```
welcome("John", "Hello")
  → Hello John!

welcome("Mary", greeting="It is nice to meet you")
   → It is nice to meet you Mary!

# this one using default value:
welcome("Paul")
  → Hi Paul!
```

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 $\longrightarrow$ one factorial

2! = 1 x 2 = 2 $\longrightarrow$ two factorial

3! = 1 x 2 x 3 = 6

4! = 1 x 2 x 3 x 4 = 24 $\longrightarrow$ four factorial

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

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

Factorial function

$1! = 1$ $\longrightarrow$ one factorial

$2! = 1 \times 2 = 2$ $\longrightarrow$ two factorial

$3! = 1 \times 2 \times 3 = 6$

$4! = 1 \times 2 \times 3 \times 4 = 24$ $\longrightarrow$ four factorial

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)
```

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
```

Useful functions: `round`

```
number = 28.301886792452830

rounded_number = round(number)
rounded_number = round(number, 1)
rounded_number = round(number, 2)
rounded_number = round(number, 3)
rounded_number = round(number, 4)
rounded_number = round(number, 5)
rounded_number = round(number, 6)
```

```
28
28.3
28.30
28.302
28.3019
28.30189
28.301887
```

Useful functions: `min` and `max`

```
num1 = 1.5
num2 = 5
num3 = 3

min_num = min(num1, num2, num3)
                                  ⟶ 1.5
max_num = max(num1, num2, num3)
                                  ⟶ 5
print("min of {0}, {1}, {2} is {3}"
      .format(num1, num2, num3, min_num))

print("max of {0}, {1}, {2} is {3}"
      .format(num1, num2, num3, max_num))
```

The `random.randint` function

import a python module called `random`

```
import random

for i in range(0, 10):
  random_number = random.randint(1, 6)
  print("Dice result: {0}".format(random_number))
```

generate a random integer between 1 and 6

```
Dice result: 3
Dice result: 2
Dice result: 4
Dice result: 1
Dice result: 3
Dice result: 1
Dice result: 3
Dice result: 1
Dice result: 6
Dice result: 5
```

generate a random integer between **`lower_bound`** and **`upper_bound`**

```
number = random.randint(lower_bound, upper_bound)
```