ENGG1811 Computing for Engineers

Week 7B: Giving functions flexibilities

Motivation

- Python functions can be made to be very flexible
- We will use the plot() function from the matplotlib library to demonstrate its flexibility
  - Python file: flexi_function.py
- The function plot():
  - Has a default behavior
  - Can accept a variable number of optional inputs, which results in different plot styles
  - The optional inputs can be specified in arbitrary order
- Goal of this lecture: How to create flexible functions?

Functions (recap)

- You saw this function in Week 3

```
def my_power(x,n):
    return x ** n

print('The value of my_power(5,2) is',my_power(5,2))

print('The value of my_power(2,5) is',my_power(2,5))
```

$x \leftarrow 5$

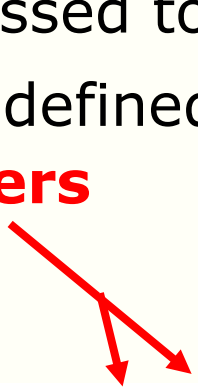
$n \leftarrow 2$

$x \leftarrow 2$

$n \leftarrow 5$

Terminology: Parameters and arguments

- Values passed to functions are called **arguments**
- Variables defined to hold these values are called **parameters**



```
def my_power(x,n):
    return x ** n

print('The value of my_power(5,2) is',my_power(5,2))

print('The value of my_power(2,5) is',my_power(2,5))
```

x ← 5
n ← 2

x ← 2
n ← 5

- Mnemonic: **a**rguments are the **a**ctual values provided to the function

A plain vanilla function

- We will use the function arrange() (in func_1.py) to show how you can make it more flexible
- The function arrange() prints two numbers in ascending or descending order

```
15 def arrange(num1, num2, order):
16     # big <- bigger number
17     # small <- smaller number
18     if num1 >= num2:
19         big, small = num1, num2
20     else:
21         big, small = num2, num1
22
23     # Print in ascending or descending
24     # order depending on the input
25     if order == 'as
26         print(order
27     elif order == '
28         print(order
29     else:
30         print('Inva
```

num1 ← 10

num2 ← 7

order ← ’ascending’

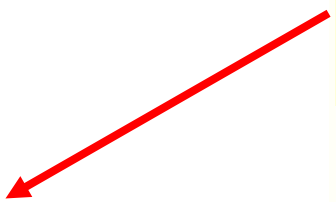
```
In [2]: arrange(10, 7, 'ascending')
ascending : 7 10

In [3]: arrange(10, 7, 'descending')
descending : 10 7
```

Keyword arguments (1)

- Python file: func_1.py

```
def arrange(num1, num2, order):
    ...

# All the following calling methods can be used
# and give the same result
arrange(10, 7, 'ascending')

arrange(10, 7, order = 'ascending')
arrange(10, num2 = 7, order = 'ascending')
```

Keyword argument



Keyword argument means you use the syntax

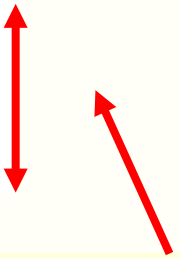
name of function parameter = argument

In the above example, *order* is the last parameter of the function *arrange()*

Keyword arguments (2)

- Python file: func_1.py

```
def arrange(num1, num2, order):
arrange(10, 7, 'ascending')

arrange(10, 7, order = 'ascending')
arrange(10, num2 = 7, order = 'ascending')
arrange(10, order = 'ascending', num2 = 7)

arrange(num1 = 10, num2 = 7 , order = 'ascending')
arrange(num2 = 7 , num1 = 10, order = 'ascending')
arrange(order = 'ascending', num2 = 7, num1 = 10)
```



Keyword arguments can be in any order.

Think about plot()

Positional arguments

- Arguments in a Python function call can either be
  - keyword arguments, or
  - positional arguments (= not keyword arguments)
- Positional arguments:
  - Are assigned to the parameter in the order they are specified

```
def arrange(num1, num2, order):
```

```
arrange(10, 7, 'ascending')

arrange(10, 7, order = 'ascending')

arrange(10, num2 = 7, order = 'ascending')
```

Mixing positional and keyword arguments

- Rule: Positional arguments cannot come after a keyword argument

```
In [11]: arrange(10, num2 = 7, 'ascending')
  File "<ipython-input-11-f8628e151fd0>", line
1
    arrange(10, num2 = 7, 'ascending')
                          ^
SyntaxError: positional argument follows
keyword argument
```

- Python assumes: Positional arguments first, then keyword arguments

✔ `arrange(10, order = 'ascending', num2 = 7)`

✖ `arrange(10, num2 = 7, 'ascending')`

- We will explain why Python imposes this restriction later

What if I don't supply all the arguments

- Open func_2_prelim.py which has arrange() as before

```
11 def arrange(num1, num2, order):
```

- Let us call it without supplying all the arguments:

```
26 arrange(2,5)
```

- The error message

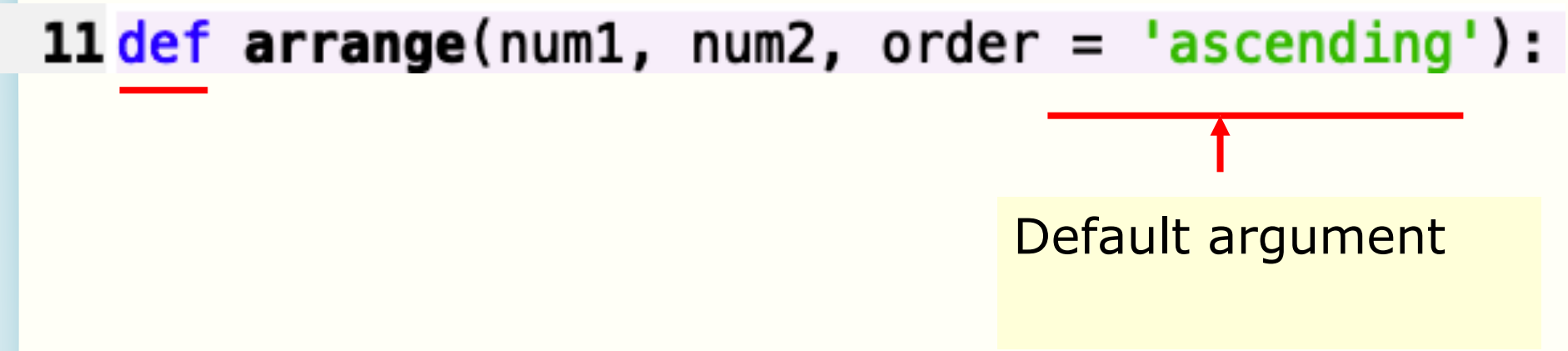
```
line 26, in <module>
    arrange(2,5)

TypeError: arrange() missing 1 required positional
argument: 'order'
```

Default argument

- We want to modify the function arrange() so that if the user has not provided the argument for the parameter *order*, then it takes on the default argument of ascending.

Default argument (1)

- Python file: func_2_prelim.py
  - Will add this part



You specify default argument at the def line of a function.

Note that it looks like an assignment, but it is NOT.

Default argument (2)

- Python file: func_2_prelim.py

```
11 def arrange(num1, num2, order = 'ascending'):
```

Default argument

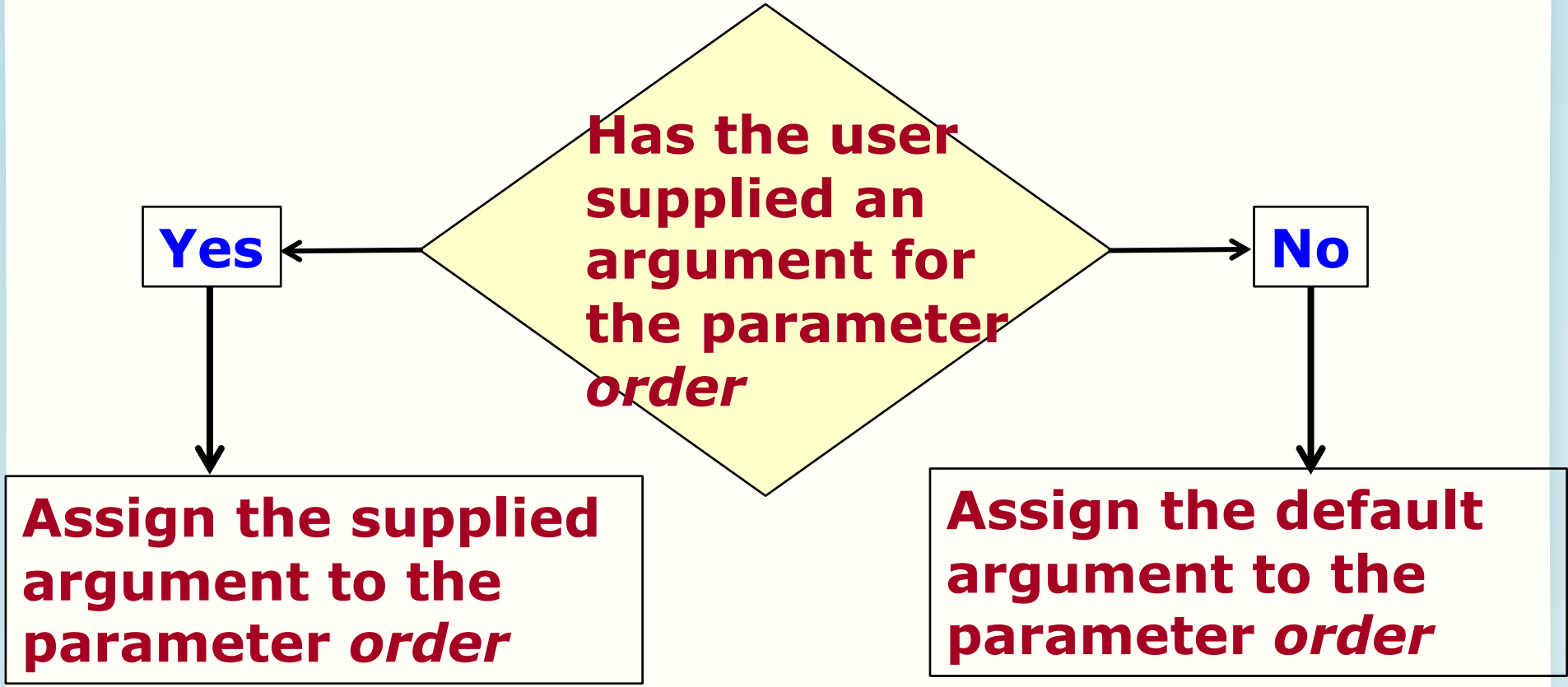
No argument has been supplied, use the default.

```
arrange(10, 7)
arrange(10, 7, 'ascending')
arrange(10, 7, order = 'ascending')
```

Same output

Behind the scene assignment

```
11 def arrange(num1, num2, order = 'ascending'):
```



Multiple default arguments

- You can have multiple default argument
- Code in file func_3.py

```
12 def arrange(num1, num2, order = 'ascending', print_orig = False):
13     # big <- bigger number
14     # small <- smaller number
15     if num1 >= num2:
16         big, small = num1, num2
17     else:
18         big, small = num2, num1
19
20     # Print in ascending or descending
21     # order depending on the input
22     if print_orig:
23         print('Original values:',num1,num2)
24
25     if order == 'ascending':
26         print(order,':',small, big)
27     elif order == 'descending':
28         print(order,':',big, small)
29     else:
30         print('Invalid option')
```

Multiple default arguments: Example (1)

- The following calling methods give the same result

```
def arrange(num1, num2, order = 'ascending', print_orig = False):
```

```
arrange(10, 7)
arrange(10, 7, 'ascending')
arrange(10, 7, 'ascending', False)
```

Multiple default arguments: Example (2)

```
12 def arrange(num1, num2, order = 'ascending', print_orig = False):
```

You want to use ascending for order and False for print_orig, so you write

```
arrange(10, 7, 'ascending', True)
```

You know ascending is the default argument for the third parameter. Can you not write that? You can't do this:

✗

```
arrange(10, 7, True)
```

But you can use keyword argument:

```
arrange(10, 7, print_orig = True)
```

Multiple default arguments

```
12 def arrange(num1, num2, order = 'ascending', print_orig = False):
```

```
arrange(10, 7, print_orig = True)
```

- The purpose of the example on the previous slide is to show you that
  - A function can have many optional inputs and each of them has a default value
  - You only have to specify the non-default inputs by using keyword arguments

Keyword arguments in action

- The following pictures come from:

https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html

```
plot([1,2,3], [1,2,3], 'go-', label='line 1', linewidth=2)
plot([1,2,3], [1,4,9], 'rs',  label='line 2')
```

```
plot(x, y, color='green', linestyle='dashed', marker='o',
     markerfacecolor='blue', markersize=12).
```

You use keyword argument to specify the options.

Because order doesn't matter, it makes it easy to use an arbitrary number of options.

Note: All positional arguments must be before the keyword arguments.

Writing functions with default arguments

Note: Python has other variations

- When writing functions
  - Put the compulsory parameters first
    - The positional arguments use these
  - After that the optional parameters with default arguments

```
def arrange(num1, num2, order = 'ascending', print_orig = False):
```

Compulsory
Positional

Optional
Default argument

- You will find the following useful tools for dealing with options
  - nested if statement
  - if/elif/else
  - Combining Boolean expressions with and/or/not

Summary

- Key concept: You can make functions flexible

- Programming concepts:
  - Terminology: parameter and argument
  - Keyword argument, positional argument
  - Default argument