Python Programming

John Dunnion

School of Computer Science
University College Dublin

Outline

Swapping two values (1)

```
# Swapping two values  #
This  doesn't work!  #
p21.py

#First  of  all, give  the  variables  values
x = 2
y = 3

print ('Before ␣swapping:')
print ('x␣is', ␣x)
print ('y␣is', ␣y)
print ()
```

Swapping two values (1)

```
# Now swap them
x = y
y = x

print('After swapping:')
print('x is', x)
print('y is', y)
```

Swapping two values (2)

- Running this program produces the following output:

```
>>>
Before swapping :
x is 2
y  is  3

After swapping :
x. is  3
y. is  3
>>>
```

- Uh oh! What happened here?

Swapping two values (3)

```
# Swapping two values #
This does work!
# p22.py

#First of all, give the variables values
x = 2
y = 3

print('Before swapping:')
print('x is', x)
print('y is', y)
print()
```

Swapping two values (3)

```
# Now swap them
temp = y
y = x
x = temp

print('After swapping:')
print('x is', x)
print('y is', y)
```

Swapping two values (4)

- Running this program produces the following output:

```
>>>
Before swapping:
x. is 2
y. is 3

After swapping:
x is 3
y is 2
>>>
```

Multiple assignment

- Python allows multiple assignment
- The statement

  x , y = 10 , 20

  assigns the value 10 to `x` and the value 20 to `y`
- All the expressions on the right-hand side of the assignment operator are evaluated before any of the assignments are carried out

Swapping two values using multiple assignment (1)

```
# Swapping two values using multiple assignment
# p23.py

x, y = 25, 36       # Give the variables values

print('Before swapping:')
print('x is', x)
print('y is', y)
print()

x, y = y, x         # Now swap them

print('After swapping:')
print('x is', x)
print('y is', y)
```

Swapping two values using multiple assignment (2)

- Running this program produces the following output:

```
>>>
Before swapping:
x. is  25
y. is  36

After swapping:
x is 36
y  is  25
>>>
```

Output revisited

- We have seen the use of the **print** function
- This produces *output*
- By default, this output goes to the “standard output” (normally the screen)

Converting Euro to Dollars (1)

```
# Converting Euro to US Dollars #
p15.py

euro_dollar_conversion = 1.12234
        # Number of US Dollars per euro #
        According to xe.com, 29.9.2016

euro_amount = 125.53        # Number of Euro

print('Conversion rate from Euro to US Dollars:',
        euro_dollar_conversion)
print('Amount in Euro:', euro_amount)

print('Amount in US Dollars:',
        euro_amount * euro_dollar_conversion)
```

Converting Euro to Dollars (2)

```
>>>
Conversion rate from Euro to US Dollars :  1.12234
Amount in Euro : 125.53
Amount in US Dollars : 140.88734019999998
>>>
```

Printing strings and variables

- Ensure that you understand the difference between

```
print ('euro_dollar_conversion')
```

and

```
print(euro_dollar_conversion)
```

- In the first case, the string "euro_dollar_conversion" is displayed on the screen
- In the second case, the value of the variable `euro_dollar_conversion` is displayed

```
euro_dollar_conversion

1.12234
```

- As we have seen, more than one word can be stored in a string variable

Another Example

Write a program to convert metres to centimetres. A simple (and fairly useless) Python program to do this is given below. This is version 1 of the program, other versions are developed as we proceed through the chapter.

```
#convert.py: converts metres to centimetres
#Author: Joe Carthy
#Date: 21/10/2022

metres = 5
centimetres = metres * 100
print("The number of centimetres is ", centimetres )
```

Executing this program produces as output:

```
% python colour.py
The number of centimetres is 500
%
```

Interactive programs

- Our programs thus far have been very *inflexible*
- Consider a Euro-Dollar conversion program that only converted e125.53 to Dollars at a rate of 1.12234!
- In order to run the program for different values, we must edit the program and re-interpret it
- Usually, we want a program to be interactive
- In other words, it should behave differently, depending on circumstances or context
- The most common example of this is trying to capture the different needs of the users, expressed by allowing them to give a different input to a program

The `input()` function

- In Python 3, there is only one function: **`input()`**
- It takes a string as an argument and displays it as a prompt (without a trailing newline) in the shell
- It then waits for the user to type something, followed by the Enter key
- The input is treated as a **string** and becomes the value returned by the function

- (The `input()` function in Python 3 has the same behaviour as the `raw_input()` function in Python 2.x)

Using `input()` (1)

```
# Greeting program
# Illustrates the use of the input() function
# p16.py

name = input('Enter your name: ')

print('Hello, ', name, '.')
```

The **input()** function displays the string and reads text from the keyboard. This text is assigned to the variable **name** in the code above

Using `input()` (2)

- Running this program produces the following output:

```
>>>
Enter your name : John
Hello , John .
>>>
```

- Note the space before the'.'

Using `input()` (3)

```
# Greeting program
#  Illustrates the use of the input ()        function
#  Uses + to prevent extra spaces  p17
#  .py

name =  input ('Enter␣your␣name: ␣')

print ('Hello ,␣' + name + '.')
          # Using  +  to  remove space  before  the  '.'
```

Using `input()` (4)

- Running this program produces the following output:

```
>>>
Enter your name : John
Hello , John .
>>>
```

- Note that there is now no space before the '.'

Another Example

```
# colour.py: Prompt use to enter colour and display a message
# Author: Joe Carthy
# Date: Oct 20 2022

favourite_colour = input('Enter your favourite colour:')

print('Yuk ! I hate ', favourite_colour)


% python3 colour.py
Enter your favourite colour: blue
Yuk ! I hate blue
%
```

Using `input()` (5)

```
# Second greeting program
# Illustrates the use of the input() function
# Uses + to prevent extra spaces #
p18.py

# First of all, get the user's name
name = input('Enter your name: ')

print('Hello, ' + name + '.')
        # Using + to remove space before the '.'

# Now get their age
age = input('What is your age? ')

print('Wow, ' + name +
      '! Your age is ' + age + '.')
```

Using `input()` (6)

```
>>>
Enter your name : John
Hello , John .
What is your age? 25
Wow, John ! Your age is 25 .
>>>
```

Using `input()` (7)

```
# Third greeting program
# Getting more chatty
# p19.py

# First of all, get the user's name
name = input('Enter your name: ')
print('Hello, ' + name + '.')
          # Using + to remove space before the '.'

# Now get their age
age = input('What is your age? ')

print('Wow, ' + name +
          '! Your age is ' + age + '.')
print('And twice your age would be ',
      age * 2, 'years!')
```

Using `input()` (8)

- Running this program produces the following output:

```
>>>
Enter your name : John
Hello , John .
What is your age? 25
Wow, John ! Your age is 25.
And twice your age would be 2525 years !
>>>
```

- Uh oh! What happened here?

Using `input()` (9)

```
# Examining the input from input () #
p20 . py

# Ask the user for an int
number = input ('Enter an int: ')

print ('Number is', number)
print ('Twice the number is', number * 2)

# Now look at the type
print (type (number))
```

Using `input()` (10)

- Running this program produces the following output:

```
>>>
Enter an int : 1234
Number is 1234
Twice the number is 12341234
<type 'str'>
>>>
```

- The `type()` function can be used to find out the type of an object

Some Types in Python

- We have already seen a number of types in Python
- `int` is used to represent integers
- Literals of type `int` are written in the way that we typically denote integers
- For example, `5`, `123`, `1000001`, `-2345`
- `float` is used to represent real (or "floating point") numbers
- Literals of type `float` include a decimal point
- For example, `1.0`, `3.1416927`, `-1234.567`
- Scientific notation can also be used: `12.34E4` represents 12.34×10^4
- `bool` is used to represent the Boolean values `True` and `False`

Type conversions

- Type conversions, or type casts, are used in Python code to convert a value to another type
- The name of the type is used to convert values to that type

```
>>> x = int('3')
>>> x * 2
6
>>> type(x)
<type 'int'>
>>>
```

- When a `float` is converted to an `int`, the number is truncated, not rounded

```
>>> int(25.9)
25
```

Using type conversion (1)

```
# Examining the input from input() #
p24.py

# Ask the user for an int
# Use a cast to make it an int
number = int(input('Enter an int: '))

print('Number is', number)
print('Twice the number is', number * 2)

# Now look at the type
print(type(number))
```

Using type conversion (2)

- Running this program produces the following output:

```
>>>
Enter an int : 1234
Number is 1234
Twice the number is 2468
<type 'int'>
>>>
```

The function **int()** converts the string from input() to a number:

```
number = int ( input ( 'Enter an int : ' ) )
```

Using type conversion (3)

Program to Convert metres to centimetres using floats.

```
#convert3.py: converts metres to centimetres
#Author: Joe Carthy
#Date: 21/10/2022

metres = float (input("Enter number of metres: "))

centimetres = metres * 100

print(metres,"metres is ",centimetres," centimetres"          )
```

```
% python convert3.py
Enter number of metres: 3.5
3.5 metres is 350.0 centimetres
%
```