Conditional statements

People are used to making decisions. For example, consider the following sentences:

If I get hungry, I will eat my lunch.

If it gets cold, I will wear my coat.

These two sentences are called **conditional sentences**. Such sentences have two parts: a **condition part** (“If I get hungry”, “If it gets cold”) and an **action part** (“I will eat my lunch”, “I will wear my coat”).

- The **action will be only be carried out if the condition is satisfied**.
- To test if the condition is satisfied we can rephrase the condition as a question with a yes or no answer.
- In the case of the first sentence, the condition may be rephrased as “Am I hungry ?” If the answer to the question is yes, then the action will be carried out (i.e. the lunch gets eaten), otherwise the action is not carried out.

- We say the condition **is true** (**evaluates to true**) in the case of a yes answer.
- We say the condition **is false** (**evaluates to false**) in the case of a no answer.
- Only **when the condition is true will we carry out the action**.
- In programming, we have the same concept. We have **conditional statements**. They operate exactly as described above.

- One of the most fundamental of these is known as the i**f statement**. This statement allows us evaluate (test) a condition and carry out an action if the condition is true.

- In Python, the keyword **if** is used for such a statement. As an example, we could modify the program to convert metres to centimetres to test if the value of metres is positive (greater than 0) before converting it to centimetres.
- The action statement(s) are indented in Python. In the program below, both if statements have action parts with 2 statements. The action statements end with the first non-indented statement follow the if.

- Note you must put a "**:**" after the condition in an if statement

```
# convert5.py: converts metres to centimetres version 3
# check quantity of metres is positive
# Outputs extra blank lines to make it easier to read the output

metres = float (input("\nEnter number of metres: "))

if metres > 0:
    centimetres = metres * 100
    print(“\n”, metres, "metres is ", centimetres, " centimetres\n\n")

if metres <= 0:
    print(“\nPlease enter a positive number for metres\n”)
    print(“\nYou entered: ”, metres \n\n")
```

Running this program:

```
Enter number of metres: -42
Please enter a positive value for metres
You entered -42
```

In this example, only one of the conditions can evaluate to true, since they are **mutually exclusive**

i.e. metres cannot be greater than 0 **and at the same time** be less than or equal to 0.

This situation arises very frequently in programming i.e.

we wish to carry out some statements when a condition is true and other statements when the **same condition** is false

A special form of the **if** statement is provided called the **if-else** statement to deal with this situation.

We rewrite the above program to illustrate its usage:

```
# convert6.py: converts metres to centimetres using else
# check quantity of metres is positive
# Outputs extra blank lines to make it easier to read the output

metres = float (input("\nEnter number of metres: "))

if metres > 0:
    centimetres = metres * 100
    print(“\n”, metres, "metres is ", centimetres, " centimetres\n\n")
else:
    print(“\nPlease enter a positive number for metres\n”)
    print(“\nYou entered: ”, metres \n\n")
```

Running this program:

```
Enter number of metres: -42
Please enter a positive value for metres
You entered -42
```

Another example: Program to calculate pay based on the number of hours worked per week.

The program below prompts the user to enter the **number of hours worked** in a week and **the rate of pay per hour.**

Workers can only work a maximum of 100 hours per week and the maximum hourly pay rate is 50.

The amount to be paid is

number of hours worked * the rate of pay per hour

The programs checks that the number of hours worked does not exceed 100 and that the rate of pay does not exceed 50

```
# pay.py: Calculate and display hourly pay

hours_worked = float(input("\nEnter number of hours worked: "))

if hours_worked > 100:
   print("\nHour worked too large:", hours_worked)
else:
   rate_per_hour = float(input("\nEnter rate per hour: "))
   if rate_per_hour > 50:
      print("\nRate per hour too high ", rate_per_hour)
   else:
      pay = rate_per_hour * hours_worked
      print("\nPay = ", pay, "for ", hours_worked, "hours")
```

Running this program:

```
Enter number of hours worked: 40

Enter rate per hour: 200

Rate per hour too high 200
```

There are only six types of condition that can arise when comparing two numbers

They can be tested for

1. **equality** - are they the same ? `metres == 0`
2. **inequality** – are they different ? `metres != 0`
3. is one **greater than** the other ? `metres > 0`
4. is one **less than** the other ? `metres < 0`
5. is one **greater than or equal to** the other ? `metres >= 0`
6. is one **less than or equal to** the other ? `metres <= 0`

Technically, the symbols ==, !=, <, >, <=, and >=, are called **relational operators**, since they are concerned with the relationship between numbers.

We call a condition metres < 0 a **Boolean expression**

This means that there are only **two possible values** (**true** or **false**) which the expression can yield.

The term **expression** is widely used in programming. Informally it means something that yields a value.

We are familiar with arithmetic expressions such as 2+2 which evaluates to 4.

A Boolean expression is one which evaluates to either true or false.

The right-hand side of an assignment statement is always an expression.

Another example, a calculator program to handle either subtraction or addition.

The user is prompted for the first number, then to enter a ‘+’ or ‘-’ character to indicate the operation to be carried out, and finally for the second number.

The program calculates and displays the appropriate result e.g.

```
Enter first number:  9
Enter operation (+ or –): –
Enter second number: 4
Taking 4.0 from 9.0 is 5.0
```

```
# calc2.py: Calculator program to add  or subtract numbers

number1 = float(input("\nEnter first number: "))

operation = input(“\nEnter operation + or –“)

number2 = float(input("\nEnter second number: "))

if operation[0] == ‘+’:
    sum = number1 + number2
    print("\n\nThe sum of",number1,"and",number2, "is", sum, "\n\n")
else:
    diff = number1 – number2
    print("\n\nTaking ",number2,"from",number1, "is", diff, "\n\n")
```

Note: `operation[0]` gives us the first element of the string entered by the user

The above programs “assumes” that if the operator is not ‘+’ then it must be ‘-‘

But the user could have hit the wrong key !

The following version checks for ‘+’, or ‘-‘ and the possibility that it was neither ‘+’ or ‘-‘

that is the user made a mistake.

User data entry mistakes are very common and good programmers always check that the user input is what was expected.

We use a third variant of **if** in the program below called *`if elif else`*

```
# calc3.py: Calculator program to add or subtract 2 numbers

number1 = float(input("\nEnter first number: "))

operation = input(“\nEnter operation + or –“)

number2 = float(input("\nEnter second number: "))

if operation[0] == ‘+’:
    sum = number1 + number2
    print("\n\nThe sum of",number1,"and",number2, "is", sum, "\n\n")
elif operation[0] == ‘–’:
    diff = number1 – number2
    print("\n\nTaking ",number2,"from",number1, "is", diff, "\n\n")
else:
    print(“\nInvalid operation only + and – allowed\n”)
    print(“You entered: “, operation[0])
```

Executing this program produces as output:

```
Enter first number:  9

Enter operation (+ or –): *

Enter second number: 4

Invalid operation – only + and – allowed
You entered: *
```

Outline

Comparison operators in Python

Python Operator	Operation
==	Equals
! =	Not equals
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Boolean operators in Python 3.x

- There are three Boolean operators: `and`, `or` and `not`
- `a and b`: If `a` is `False`, it returns `a`, otherwise it returns `b`
- `a or b`: If `a` is `False`, it returns `b`, otherwise it returns `a`
- `not a`: If `a` is `False`, it returns `True`, otherwise it returns `False`

Python Operator	Operation
`not`	Logical NOT
`and`	Logical AND
`or`	Logical OR

Using Boolean operators in Python

```
>>> a = 2
>>> b = 3
>>> c = 10
>>> d = 10
>>> a < b
True
>>> c > b
True
>>> c < d
False
>>> d == d
True
>>> c == d
True
>>> c != d
False
```

Conditions

- We are familiar with making decisions based on conditions
- *If I am hungry, I will eat my dinner*
- *If I am cold, I will put on my coat*
- *If the number is even, I will divide the number by 2*
- Such sentences are called conditional sentences
- Such sentences have two parts:
  - A condition or test:
    If I am hungry, *If I am cold*, *If the number is even*
  - An action:
    I will eat my dinner, *I will put on my coat*, *I will divide the number by 2*
- The action will only be carried out if the condition is satisfied (or the test is true)
- Optionally, there is another action that will be carried out if the condition is not satisfied (or the test is false)

Sequential statements

- The programs that we have seen so far have contained only sequential statements
- Such programs follow a sequential flow of control
- There is a single execution path through the program
- These can be called *straight-line programs*
- In such a program, statements are executed in the order in which they appear
- The program stops when control reaches the final statement
- The type of problem that we can solve with such a program is very simple and very limited

Conditional statements

- Most programming languages allow for programs that have more than one execution path through them
- Such programs follow a conditional flow of control
- These can be called *branching programs*
- A conditional statement has two or three parts:
  - Optionally, a statement, or block of statements, that is executed when the condition evaluates to `False`
  - A test, ie an expression that evaluates to either `True` or `False`
  - A statement, or block of statements, that is executed when the condition evaluates to `True`
- After the conditional statement, execution resumes at the statement following the conditional statement

Conditional statements

- Conditional statements allow us to change the *flow of control* in a program
- Within a program, a condition can be tested and actions carried out only if the condition is `True`
- This gives programs much more power and flexibility

Conditional statement in Python (1)

- In Python, a conditional statement has one of the following forms:
- **if** *Boolean expression*:
  statement(s)
- **if** *Boolean expression*:
  statement(s)
  else:
  statement(s)
- **if** *Boolean expression*:
  statement(s)
  elif *Boolean expression*:
  statement(s)
  else:
  statement(s)

Conditional statement in Python (2)

- In describing the forms of the conditional statement, italics are used to describe the type of Python code that can occur at that point in the statement
- *Boolean expression* indicates that any expression that evaluates to `True` or `False` can follow the reserved words `if` or `elif`
- *statement(s)* indicates that any sequence of Python statements can appear at those points

Using the conditional statement in Python (1)

- Consider the following program that prints “Number is zero” if the number entered by the user is 0

```
# Using the conditional statement
# Prints 'Number is zero' if the number
        entered is 0
# p25.py

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

if number == 0:
    print('Number is zero')
print('Finished!')
```

Using the conditional statement in Python (2)

- Example outputs from this program are the following:

```
>>>
Enter an int: 123
Finished!
>>>
>>>
Enter an int: 0
Number is zero
Finished!
>>>
>>>
Enter an int: -5
Finished!
>>>
```

Evaluating the Boolean expression

- The expression `number % 2 == 0` evaluates to `True` when the remainder of `number` divided by 2 is 0, and evaluates to `False` otherwise
- Recall that `==` is the operator used for comparison
- The `=` operator is used only for assignment
- A number is even (2, 4, 6, 8, …) if it is divisible by 2
- A number is divisible by 2 if

```
number % 2 == 0
```

evaluates to `True`

Using the conditional statement in Python (3)

- Consider the following program that tests the number entered by the user and prints “Number is even” or “Number is odd”

```
# Using the conditional statement
# Prints 'Number is even' or
# 'Number is odd'
# p26.py

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

if number % 2 == 0:
    print('Number is even')
else:
    print('Number is odd')
print('Finished!')
```

Using the conditional statement in Python (4)

- Example outputs from this program are the following:

```
>>>
Enter an int: 3
Number is odd
Finished!
>>>
>>>
Enter an int: 2424
Number is even
Finished!
>>>
```

Indentation

- Note that indentation is significant in Python
- Statements at the **same level of indentation** belong to the same block of statements
- Different languages use different mechanisms to mark blocks of statements
- For example, Pascal uses `begin` and `end` keywords
- C and Java use braces (curly brackets), ie `{` and `}`
- Some languages use the keyword that introduces the block spelled backwards, eg `if` and `fi`
- Python is unusual in using indentation in this way.
- Programs *should* be indented
- Python's indentation forces the programmer to indent their programs properly and in a standard way

Currency Conversion Program: Algorithm

- Consider a more sophisticated program to convert Euro to Dollars
- We only want to convert Euro amounts that are greater than zero
- We start off by writing an algorithm for this program

Prompt the user for a Euro amount
Read the Euro amount
if *the Euro amount ≥ 0* ***then***
    Perform the conversion
    Print out the Dollar amount
else
    Tell the user that the amount must be ≥ 0
Program finishes

Currency Conversion Program: Program

```
# Converting Euro to US Dollars
# p27.py

rate = 1.117    # 1 euro = 1.117 usd


# Ask the user to enter the Euro amount
euro_amount = int(input('Enter amount of Euro: '))
print('Amount in Euro:', euro_amount)

if euro_amount >= 0:
    print('Amount in Dollars:',euro_amount * rate)
else:
    print('Amount must be >= 0.')
    print('Please try again.')

print('Finished!')
```

Currency Conversion Program: Output

- Example outputs from this program are the following:

```
Enter the amount of Euro you wish to convert : 1000
Amount in Euro : 1000
Amount in US Dollars : 1117.0000
 Finished !
>>> ================================ RESTART ===
Enter the amount of Euro you wish to convert : 0
Amount in Euro : 0
Amount in US Dollars : 0.0
Finished !
>>> ================================ RESTART ===
Enter the amount of Euro you wish to convert : -1
Amount in Euro : -1
Amount must be >= 0.
Please try again.
Finished !
>>>
```

Conditional statement (1)

```
if num > 0:
    print('Number is positive.')
elif num == 0:
    print('Number is equal to 0')
else:
    print('Number is negative.')
```

Conditional statement (2)

```
if num == 0:
    print('Number is equal to 0')
elif num > 0:
    print('Number is positive.')
else:
    print('Number is negative.')
```

Conditional statement (3)

```
if num > 0:
    print('Number is positive.')
elif num == 0:
    print('Number is equal to 0')
elif num < 0:
    print('Number is negative.')
```

Boolean conditions (1)

- We have already seen the three Boolean operators: `and`, `or` and `not`
- These can be used to create complex Boolean conditions

```
if num_hours < 0 or num_hours > 168:
    print('Number of hours worked per week shoul
      positive and be a maximum of 168!')
```

Boolean conditions (2)

- Consider the following:

```
if num > 20:
    if num % 2 == 0:
        print('Number is even and greater than 20')
```

- and

```
if num > 20 and num % 2 == 0:
    print('Number is even and greater than 20')
```

Boolean conditions (3)

- Consider the following:

```
if num > 20 and num % 2 == 0:
    print('Number is even and greater than 20')
```

- and below which does NOT behave same as above

```
if num > 20 or num % 2 == 0:
    print('Number is even and greater than 20')
```

The second one is incorrect – it will execute the print if either num > 20; or if num % 2 == 0

Boolean conditions (4)

- Consider the following:

```
if num_hours < 0 or num_hours > 168:
    print('Number of hours worked per week '
    'should be positive and be a maximum of 168!')
```

- and incorrectly

```
if num_hours < 0 and num_hours > 168:
    print('Number of hours worked per week '
    'should be positive and be a maximum of 168!')
```

The second print will never be executed because num_hours cannot be BOTH < 0 and > 168 at the same time.

Letters

The **uppercase** letters are A, B, C , … to Z

The **lowercase** letters are a, b, c, …. to z

We can check if a variable `letter` is uppercase by the test

`If letter >= 'A' and letter <= 'Z'` then the letter must be in the range from A to Z

Similarly to test for lowercase
`If letter >= 'a' and letter <= 'z'` then the letter must be in the range from a to z

- Consider the following:

```
s = input(”\n Enter y or Y: “)
if   (s[0] == ‘y’)  or   (s[0] == ‘Y’):
  print(‘\nYou entered y or Y’)


s = input(”\n Enter a letter a to z: “)
if   (s[0] < ‘a’)  or   (s[0] > ‘z’):
  print(‘\n Not a lowercase letter’, s[0])

s = input(”\n Enter a letter A to Z: “)
if   (s[0] < ‘A’)  or   (s[0] > ‘Z’):
  print(‘\n Not an uppercase letter’, s[0])
```

```
s = input("\n Enter a letter a to z: ")

if   (s[0] >= 'a')  and   (s[0] <= 'z'):
  print('\n Yes - lowercase letter', s[0])


s = input("\n Enter a letter A to Z: ")

if   (s[0] >= 'A')  and   (s[0] <= 'Z'):
  print('\n Yes uppercase letter', s[0])
```