



Technical University of Lodz
Institute of Electronics

Algorithms and Data Structures

4. Loops and Boolean Expressions

Łódź 2012





Exercise – Harmonic Sum

- Type in the program in *Notepad++*
- Save it as **harmonic.py**
- Run the script using *Pylab*

```
1  # harmonic.py
2
3  def harmonic(n):
4      # Compute the sum of 1/k for every k from 1 to n.
5      total = 0.0
6      for k in range(1, n + 1):
7          total += 1.0 / k
8      return total
9
10 def main():
11     n = input('Enter a positive integer: ')
12     print "The sum of 1/k for k = 1 to %d is %.6f" % (n, harmonic(n))
13
14 main()
15
```

- This program uses the **for** loop, the **range()** function and **+=** operator.



for Loops

- The **for** loop is used to repeat the calculations in cases one knows ahead of time how many times the loop needs to run.

Syntax:

```
for <variable> in <sequence>:  
    <body>
```

- For each item in the **<sequence>**, the **<variable>** is assigned the value of that item and the **<body>** is executed.

- A **sequence** in Python is an ordered set of elements. An example is the **list** whose elements are listed in square brackets, e.g. [12, -3, 5, 0.5].

```
>>> for x in [12, -3, 5, 0.5]:  
>>>     print ( x, x**2)
```

- What is the body of the above loop? How many times a body is executed in the general case?



range Function

- This is a built-in function that creates a sequence of integers.

Syntax:

range(stop)	Begin at 0 . Take steps of 1 . End just before stop .
range(start, stop)	Begin at start . Take steps of 1 . End just before stop .
range(start, stop, step)	Begin at start . Take steps of step . End just before stop .

- The variables **start**, **stop** and **step** are integers.

```
>>> range(9)
>>> range(-2, 12)
>>> x = 5
>>> range(3, x+7)
>>> range(3, x-2)
>>> range(3, x-1)
>>> range(5, 2, -1)
```



Accumulation Loops

- Script *harmonic.py* contains an accumulator variable **total** which gradually accumulates some quantities as the loop runs.

Syntax:

```
<accumulator> = <starting value>
loop:
    <accumulator> += <value to add>
```

- The **x += v** is a *shorthand assignment* equivalent to **x = x + v**
- Accumulation loops may accumulate also through other operations, e.g. subtraction, multiplication and division (**x -= v**, **x *= v**, **x /= v**) rather than addition.



Do not use **sum** as an accumulator variable name, because it is a built-in Python function.

```
>>> x = range(6)
>>> print ( sum ( x ) )
```

- For many repetitions, the program may run much too long. To break the loop, use **CNTRL-C** or restart Pylab or Python shell.



Body Mass Index Revisited

- Make the script using *Notepad++*, save as *bmi_evaluation.py* and run in *Pylab*.

```
1 # bmi_evaluation.py
2
3 YourName = input( "Enter your name: " )
4 var = input ( "Enter your height [in cm]: " )
5 YourHeight = var / 100.0
6 YourMass = input ( "Enter the mass of your body [in kg]: " )
7 BMI = YourMass / ( YourHeight * YourHeight)
8 print ("\n")
9 print ("Hello " + YourName + "!")
10 if (BMI < 18.5):
11     print "Your BMI is %4.1f (possible underweight)." % ( BMI )
12 elif (BMI >= 18.5) and (BMI < 25):
13     print "Your BMI is %4.1f (correct weight)." % ( BMI )
14 else:
15     print "Your BMI is %4.1f (possible overweight)." % ( BMI )
```



Boolean (Logic) Expressions

- Another type of data in Python is **boolean**. Variables that hold this type have only 2 possible values: **True** or **False**.
- Python has the following **comparison operations** that return boolean values:

<code>x == y</code>	Equal.
<code>x != y</code>	Not equal.
<code>x < y</code>	Less than.
<code>x > y</code>	Greater than.
<code>x <= y</code>	Less than or equal.
<code>x >= y</code>	Greater than or equal.



Avoid using `==` or `!=` to compare floats!

```
>>> x = 3
>>> y = 2
>>> P = x == y #P is a Boolean variable
>>> print (P)
```



Boolean (Logic) Expressions

- Boolean expressions may be combined with this **boolean operations**:

P and Q	True if both P and Q are True ; otherwise False .
P or Q	True if either P or Q are True ; otherwise False .
not P	True if P is False ; otherwise False .

```
>>> x = 3
>>> y = 2
>>> P = not ((x == y) or (y == 2))
>>> print (P)
```

```
>>> P = True
>>> Q = False
>>> R = P and Q
>>> print R
```




if Statement

Perhaps all programming languages have some form of **if** statement.
In Python, it has essentially 3 forms.

```
if <boolean> :  
    <body>
```

```
if <boolean> :  
    <body1>  
else:  
    <body2>
```

```
if <boolean1> :  
    <body1>  
elif <boolean2> :  
    <body2>  
elif <boolean3> :  
    <body3>  
...  
else :  
    <bodyN>
```

```
>>> YourGender = 'Female'  
>>> if YourGender == 'Male':  
>>>     print('Good day, Mister!')  
>>> else:  
>>>     print('Nice to see you, Madame!')
```



while Loops

- The **while** loops allow program to repeat the calculations without necessarily knowing in advance how many times the loop will run.

Syntax:

```
while <boolean> :  
    <body>
```

-The **<boolean>** expression is evaluated, and if it is **True**, then the **<body>** is executed. After that, the **<boolean>** is checked again and if it is still **True**, the **<body>** executes again. This is repeated until the boolean expression is **False**.

(a) Print all positive integers up to n

```
>>> n = 5  
>>> i = 1  
>>> while i < n+1:  
>>>     print i  
>>>     i += 1
```

(b) Print even numbers up to n

```
>>> n = 10  
>>> i = 1  
>>> while i < n+2:  
>>>     if i%2 == 0: print i  
>>>     i += 1
```



Temperature Converter

- In the following **temp.py** program the while loop repeats when user enters an invalid temperature value.

```
1  # temp.py
2  # The entered temperature in Celsius
3  #   is converted to Fahrenheit degrees.
4  temp = input ( 'Enter a temperature in Celsius: ')
5  while temp<-273.15:
6      temp = input ('Impossible. Enter a valid temperature: ')
7  print "In Fahrenheit, that is", 9.0/5.0*temp+32.0
8  |
```

- Unlike the **for** loop, it is possible for a **while** loop to be **infinite**. Press CTRL-C to break calculations of an infinite loop.

```
>>> i = 0
>>> while True:
>>>     print(i)
>>>     i += 1
```



Prime numbers

- In the following **prime.py** program, the **while** loop is used to find out whether an entered number is prime or not.

```
1  # prime.py
2
3  num = input ( 'Enter an integer number: ')
4  i = 2
5  # A prime is a number that divides by 1 and itself only.
6  while i<num and num%i != 0: # num%i equals to remainder
7      i += 1
8  if i==num:
9      print "The number", num, "is prime."
10 else:
11     print "The number", num, "is not prime."
12
```



Prime numbers – tracing the code

```
>>> i = 2
>>> while i < num and num % i != 0:
>>>     i += 1
```

 Loop terminated

num = 9:

<i>i</i>	<i>i < num</i>	<i>num % i</i>	Boolean expression
2	T	1	T
3	T	0	F
-	-	-	-
-	-	-	-

num = 5:

<i>i</i>	<i>i < num</i>	<i>num % i</i>	Boolean expression
2	T	1	T
3	T	2	T
4	T	1	T
5	F	-	F

On termination: *i = 3*
i < num → not prime

i = 5
i = num → prime



Exercises

4.1. Explain the use of $n+1$ instead of n in **harmonic.py**. Conjecture what happens to the harmonic sum for very large n .

4.2. Assign the value of 10 to a variable n ($n = 5$), run the **harmonic.py** program and enter 3 when asked for a positive integer that is assigned to the variable n in the program. Then print the value of n and explain the result. List all the local variables and describe the scope of each.

4.3. Determine a **range** expression to iterate over each of these sequences

(a) 0, 1, 2, 3

(b) 3, 2, 1, 0

(c) 1, 3, 5, 7, 9, 11

(d) 10, 27, 44, 61, ..., 197

(e) 100, 90, ..., 10

(f) 2, 4, 6, 8, ..., 200

4.4. Write a program **myfactorial.py** that returns the product $1*2*3*...n$. Use an accumulator; do not use the **factorial()** function from the **math** module.



Exercises

- 4.5. Write a **mymax.py** function that returns the largest of x , y and z .
- 4.6. Write a **median3.py** function that returns the middle value among the x , y and z .
- 4.7. Write a **myabs.py** function that returns the absolute value of x .
- 4.8. Find the smallest number in the list.
- 4.9. Find the largest number in the list.
- 4.10. Write a Python code example of a nested loop.
- 4.11. Write an infinite loop.



Summary

- 1) The **for** loop is used when one knows in advance how many times calculations are to be repeated.
- 2) Python **list** stores an ordered sequence of items. The items (list elements) can be of any type, e.g. `a_list = [10, False, "Hi mate!", 3.14159]`.
- 3) Python **boolean** data type can have any of two values: **True** or **False**.
- 4) Boolean values are returned by **comparison operations**.
- 5) The function **range()** produces a list of integers.
- 6) The **if statement** allows doing different parts of a program provided some conditions are satisfied.
- 7) The **while** loop is used to program repetitive computations without knowing the number of repetitions in advance.
- 8) Loops can sometimes be infinite.



Literature

Brian Heinold, Introduction to Programming Using Python, Mount St. Mary's University, 2012 (<http://faculty.msmary.edu/heinold/python.html>).

Brad Dayley, Python Phrasebook: Essential Code and Commands, SAMS Publishing, 2007 (dostępne też tłumaczenie: B. Dayley, Python. Rozmówki, Helion, 2007).

Mark J. Johnson, A Concise Introduction to Programming in Python, CRC Press, 2012.