THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 4
Variables and simple loops

Revision: functions and module file

```
# File: cel2fah.py
# A simple program is illustrating Celsius to Fahrenheit conversion

def c2f():
    print("This program converts Celsius into Fahrenheit")
    cel = float(input("Enter temperature in Celsius: "))
    fah = 9/5*cel+32
    print("The temperature in Fahrenheit is:",fah)
    print("End of the program.")
c2f()
```

- We use a filename ending in .py when we save our work to indicate it's a Python program.
- Click green button (run) on Thonny to run the program.

Objectives of this Lecture

- To understand the process of assigning values to variables
- To look into limitations of data types
- To understand simple loops
- Follow the software development process to develop a program that calculates the future value of investment

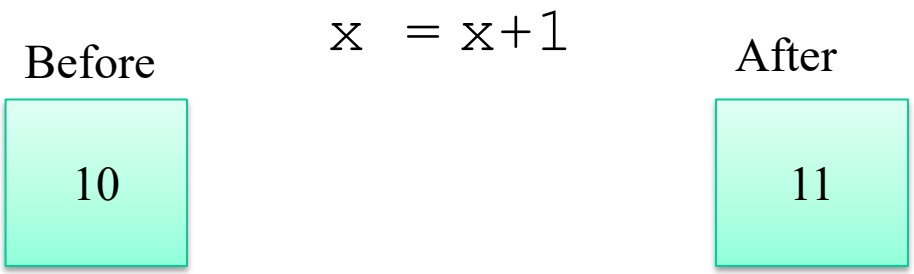
Assignment Statements

- Simple Assignment
- <variable> = <expr>
  variable is an identifier, expr is an expression
- The expression on the right is evaluated to produce a value which is then associated with the variable named on the left.

Assignment Statements – the Simple View

- Variables are like a box we can put values in.

- When a variable changes, the old value is erased and a new one is written in.



Assignment Statements

- `x = 3.9 * x * (1-x)`
- `fahrenheit = 9/5 * celsius + 32`
- `x = 5`

The spacing around parts of assignments is optional, but makes the resulting program much more readable – and hence maintainable.

Assignment Statements

- Variables can be reassigned as many times as you want!

```
>>> myVar = 0
>>> myVar
0
>>> myVar = 7
>>> myVar
7
>>> myVar = myVar + 1
>>> myVar
8
```

Simultaneous Assignment

- Several values can be calculated at the same time
- <var>, <var>, … = <expr>, <expr>, …
- Evaluate the expressions on the right and assign them to the variables on the left
  - *Must have same number of expressions as variables!*

```
>>>sum, diff = x+y, x-y
```

Simultaneous Assignment

- How could you swap the values for x and y?
  - *Why doesn't this work?*

```
x = y
y = x
```

  - *#assume x= 1, y= 9 initially….. x= y will make x= 9 (original x is lost)*
- We could use a temporary variable…

```
temp = x
x = y
y = temp
```

Simultaneous Assignment

We can swap the values of two variables easily in Python!

```
>>> x = 3
>>> y = 4
>>> print(x, y)
3 4
>>> x, y = y, x
>>> print(x, y)
4 3
```

Increment Assignment

- Certain types of assignment statement are so common that a short-cut exists

  `x = x + n,` (especially `x = x +1`)

  `x = x - n` (n can be any expression)

- These become

  `x += n`

  `x -= n`

- `Also`

  `x *= n`

  `x /= n`, *etc*

Numeric Data Types

- There are two different kinds of numbers!
  - *5, 4, 3, 6 are whole numbers – they don't have a fractional part*
  - *0.25, 1.10, 3.142 are decimal fractions*

- Inside the computer, whole numbers and decimal fractions are represented quite differently!
  - *We say that decimal fractions and whole numbers are two different data types.*

Numeric Data Types

- The data type of an object/variable determines
  - *what values it can have*
  - *and what operations can be performed on it*
  - *Taking 3 from 10: easy*
  - *Taking 3 from J: ?*

Numeric Data Types

- Whole numbers are represented using the *integer* (*int* for short) data type.
  - *Size depends on machine using it*

- *long* is another data type similar to *int* but can store longer number
  - *Needs more memory space*

- These values can be positive or negative whole numbers.

Numeric Data Types

- Numbers that can have fractional parts are represented as *floating point* (or *float*) values.

- How can we tell which is which?

  – *A numeric literal without a decimal point produces an int value*

  – *A literal that has a decimal point is represented by a float (even if the fractional part is 0)*

Numeric Data Types

- Why do we need two number types?
  - *Values that represent counts can't be fractional*
  - *Most mathematical algorithms are very efficient with integers*
  - *The float type stores only an approximation to the real number being represented!*
  - *Since floats aren't exact, use an int whenever possible!*

Numeric Data Types

- Operations on ints produce ints (excluding /)
- Operations on floats produce floats.

```
>>> 3.0+4.0
7.0
>>> 3+4
7
>>> 10.0/3.0
3.3333333333333335
>>> 10/3
3.3333333333333335
>>> 10 // 3
3
```

/ does floating point division

// does integer division

Numeric Data Types

- Integer division produces a whole number.
  - *That's why* `10//3 == 3`

- Think of it as 'goes into', where 10//3 == 3 since 3 (goes into) 10, three times (with a remainder of 1)

- 10 % 3 = 1 is the remainder of the integer division of 10 by 3.

Limits of Int

- What's going on?
  - *While there are an infinite number of integers, there is a finite range of integers that can be represented by* int.
  - *This range depends on the number of bits a particular CPU uses to represent an integer value.*

Limits of Int

- Typical PCs use 64 bit integers.
- That means there are 2^{64} possible values, centered at 0.
- This range is -2^{63} to $2^{63}-1$. We need to subtract one from the top end to account for 0.
- Python solution (recent Python versions): expanding int

- Does switching to *float* data types get us around the limitations of *int*?

```
>>> x1 = 10**121
10000…
>>> y1 = x1 + 1
10000… … … 1
>>> z1 = x1 - y1
-1
>>> x2 = 10e121
1e+122
>>> y2 = x2 + 1
1e+122
>>> z2 = x2 - y2
0.0
```

Handling Large Integers

- Floats are approximations
- Floats allow us to represent a larger range of values, but with fixed precision.
- Python int is not a fixed size, but expands to handle whatever value it holds.
- Newer versions of Python automatically convert int to an expanded form when it grows so large as to overflow.
- Can store and work with indefinitely large values (e.g. 100!) at the cost of speed and memory

Type Conversion

- Combining an `int` with a `float` in an expression will return a `float`
- We can explicitly convert between different data types
- For example the `int` and `round` functions convert a float to integer

```
>>> int(6.8)    # Truncate
6
>>> round(6.8)
7
>>> float(6)
6.0
```

Type Conversion : More examples

Conversion function	Example use	Value returned
int(<a number or string>)	int(2.87)	2
int	int(“55”)	55
float(<a number or string>)	float(32)	32.0
str(<any value>)	str(43)	‘43’

str: string/text

Find Data Type

- We can use the type function to find the data type

```
>>> type(4)
<class 'int'>

>>> type(4.3)
<class 'float'>

>>> x = 5.76
>>> type(x)
<class 'float'>

>>> type(hello) # Without quotes, assumes 'hello'  is a
                # variable. Not defined, so will generate
                # an error
>>> type('hello')
<class 'str'>
```

If a variable exists, the type of the variable is the type of the value assigned to it

Scientific Notation

Decimal Notation	Scientific Notation	Meaning
2.75	2.75e0	2.75×10^{0}
27.5	2.75e1	2.75×10^{1}
2750.0	2.75e3	2.75×10^{3}
0.0275	2.75e-2	2.75×10^{-2}

- Python floating point values go from -10^{308} to 10^{308}
- Typical precision is 16 digits (decimal places)

Float Problems

- Very large and very small floating point values can also cause problems (current Python)

```
>>> x = 1.0e308
>>> x
1e+308
>>> x = 100* x
>>> x
inf
```

This is called
over-flow

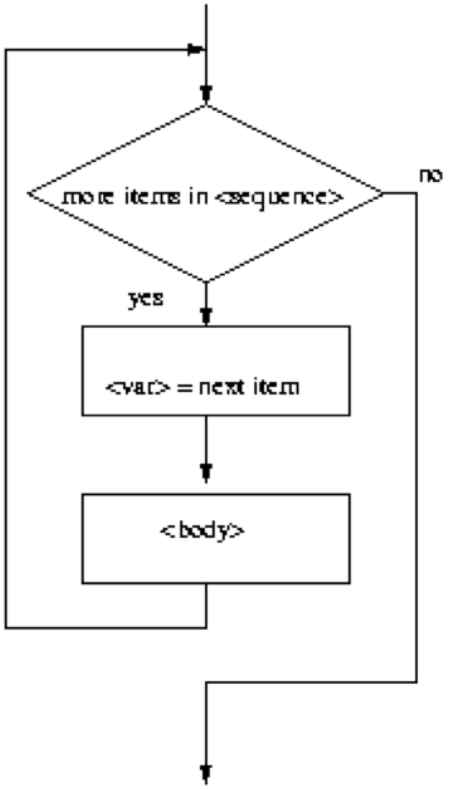
```
>>> x = 1.0e-323
>>> x
1e-323
>>> x = x / 100.0
>>> x
0.0
```

This is called
under-flow

Simple loops

Definite Loops

`for` loops alter the flow of program execution, so they are referred to as control structures.



Definite Loops

- A definite loop executes a pre-specified number of times, iterations, which is known when program loaded
- for <var> in <sequence>:

  <body>
- The beginning and end of the body are indicated by indentation.
- Note that iterations are over sequences (more of this in a later lecture)
  - *For the time being, a sequence is a countable sequence of Python things (objects)*

Definite Loops

- `for <var> in <sequence>:`

  `<body>`

- The variable after the `for` is called the loop index. It takes on each successive value in sequence

Definite Loops

- In `chaos.py` (from Labsheet 00), what did `range(10)` do?

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- `range` is a built-in Python function that generates a sequence of numbers, starting with 0.
- `list` is a built-in Python function that turns the sequence into an explicit list
- The body of the loop executes 10 times

Definite Loops

```
>>> for odd in [1, 3, 5, 7]:
        print(odd*odd)
1
9
25
49
```

Loop variable `odd` first has the value 1, then 3, then 5 and finally 7

Example Program: Interest Earned

Analysis

- *Money deposited in a bank account earns interest.*
- *How much will the account be worth 10 years from now?*
- *Inputs: principal amount, interest rate*
- *Outputs: value of the investment in 10 years*

Example Program: Future Value

Specification

- **Inputs**

  principal The amount of money being invested, in dollars

  apr The *annual percentage rate* expressed as a floating point decimal number 0.0 < `apr` < 1.0

- **Output**

  The value of the investment 10 years in the future

- **Relationship**

  Value after one year is given by *principal* * (1 + *apr*).

  This needs to be done 10 times.

Example Program: Future Value

Design

- *Print an introduction*
- *Input the amount of the principal (`principal`)*
- *Input the annual percentage rate (`apr`)*
- *Repeat 10 times:*
  *principal = principal * (1 + apr)*
- *Output the value of principal*

Example Program: Future Value

Implementation

- *Each line translates to one line of Python (in this case)*
- *Print an introduction*

```
print("This program calculates the future")
print("value of a 10-year investment.")
```

- *Input the amount of the principal*

```
principal = float(input("Enter the initial principal: "))
```

Example Program: Future Value

- *Input the annual percentage rate*

```
apr = float(input("Enter the annual interest rate: "))
```

- *Repeat 10 times:*

```
for i in range(10):
```

- *Calculate principal = principal * (1 + apr)*

```
    principal *= (1 + apr)
```

- *Output the value of the principal at the end of 10 years*

```
print("The value in 10 years is:", principal)
```

Example Program: futval.py

```
#    A program to compute the value of an investment
#    carried 10 years into the future
#    Author: Unit Coordinator

def main():
   print("This program calculates the future“)
   print("value of a 10-year investment.")

   principal = float(input("Enter the initial principal: "))
   apr = float(input("Enter the annual interest rate: "))

   for i in range(10):
       principal *= (1 + apr)

    print ("The value in 10 years is:", principal)

main()
```

Example Program: Testing futval.py

```
>>> main()
This program calculates the future value of a 10-year investment.
Enter the initial principal: 100
Enter the annual interest rate: 0.03
The value in 10 years is: 134.391637934
>>> main()
This program calculates the future value of a 10-year investment.
Enter the initial principal: 100
Enter the annual interest rate: 0.10
The value in 10 years is: 259.37424601
```

Lecture Summary

- Understanding the concept of assignment in Python

- We learned how to
  - *assign values to variables*
  - *do multiple assignments in one statement*
  - *definite simple definite loops*
  - *Limitations of data types*

- "for" loop alters the sequence of the program