COMPSCI 101

Principles of Programming

Lecture 2 - Variables, program execution, doing calculations, print()

Learning outcomes

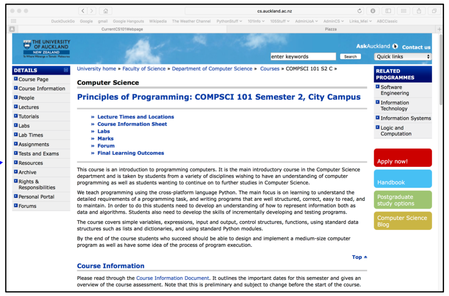
At the end of this lecture, students should be able to:

- perform calculations using standard arithmetic operators
- use variables to store values
- describe differences between int and float types
- print numbers and strings to standard output

Installing Python 3

Go to the resources page of the CompSci 101 website. You will see the link to python.org where you will be able to download the python installer. Make sure you install **Python 3**.

Resources →



University home » Faculty of Science » Department of Computer Science » Courses » COMPSCI 101 S2 C »

Computer Science

Principles of Programming: COMPSCI 101 Semester 2, City Campus

» Lecture Times and Locations
» Course Information Sheet
» Labs
» Marks
» Forum
» Final Learning Outcomes

This course is an introduction to programming computers. It is the main introductory course in the Computer Science department and is taken by students from a variety of disciplines wishing to have an understanding of computer programming as well as students wanting to continue on to further studies in Computer Science.

We teach programming using the cross-platform language Python. The main focus is on learning to understand the detailed requirements of a programming task, and writing programs that are well structured, correct, easy to read, and to maintain. In order to do this students need to develop an understanding of how to represent information both as data and algorithms. Students also need to develop the skills of incrementally developing and testing programs.

The course covers simple variables, expressions, input and output, control structures, functions, using standard data structures such as lists and dictionaries, and using standard Python modules.

By the end of the course students who succeed should be able to design and implement a medium-size computer program as well as have some idea of the process of program execution.

Course Information

Please read through the Course Information Document. It outlines the important dates for this semester and gives an overview of the course assessment. Note that this is preliminary and subject to change before the start of the course.

https://www.cs.auckland.ac.nz/courses/compsci101s2c/resources/

A program is a sequence of instructions

A program is a sequence of instructions which performs a specific task

- Instructions are specified in a sequence
- Computers execute the instructions one after the other

Instructions are specified using a formal language

- Natural languages are the languages spoken by people
- Formal languages are designed by people for a specific purpose, e.g., mathematical notation, chemical structure of molecules, programming languages

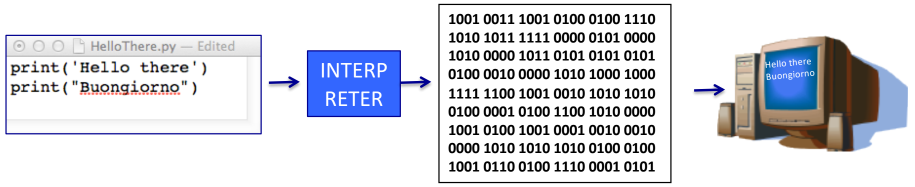
We shall be writing our programs in the **Python 3** programming language

The Python interpreter

Source code (programs) is written in a programming language such as Python.

The Python **interpreter** translates and **executes** source code

- One instruction at a time is converted into machine code and executed by the interpreter.



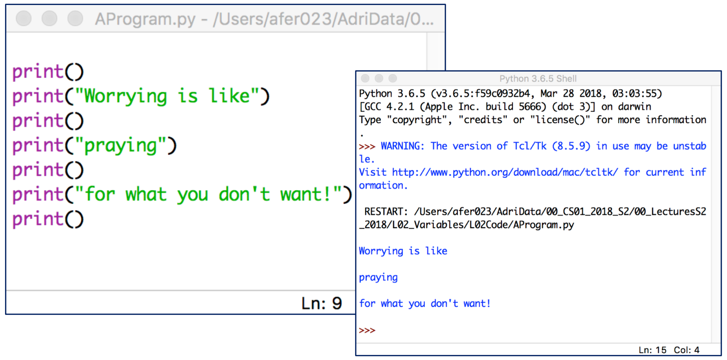
```
print('Hello there')
print("Buongiorno")
```

```
1001 0011 1001 0100 0100 1110
1010 1011 1111 0000 0101 0000
1010 0000 1011 0101 0101 0101
0100 0010 0000 1010 1000 1000
1111 1100 1001 0010 1010 1010
0100 0001 0100 1100 1010 0000
1001 0100 1001 0001 0010 0010
0000 1010 1010 1010 0100 0100
1001 0110 0100 1110 0001 0101
```

IDLE – The program editor used in CompSci 101

IDLE (**I**ntegrated **D**eve**L**opment **E**nvironment) is an integrated development environment for Python. This is the development environment provided when you download Python.

This is the environment we will be using to write and execute our Python programs.



```
print()
print("Worrying is like")
print()
print("praying")
print()
print("for what you don't want!")
print()
```

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 03:03:55)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information
.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstab
le.
Visit http://www.python.org/download/mac/tcltk/ for current inf
ormation.

 RESTART: /Users/afer023/AdriData/00_CS01_2018_S2/00_LecturesS2
_2018/L02_Variables/L02Code/AProgram.py

Worrying is like

praying

for what you don't want!

>>>
```

Programs are deterministic

- the result of the program instructions is well defined,
- rules govern the results of instructions. Once we learn the rules, we can govern what the computer does,
- the output is completely predictable

Storing information - variables

Variables are names for storage locations

- Almost any name will work – **but there are some constraints**
- A variable stores only one value at a time
- Assign a value to a variable location using **=** (the **assignment operator**)
- Refer to the value in a location using the variable name.

Three variables used to store three pieces of information.

```
name = "Adriana"
height = 170.0
age = 21
```

Variable names

The following are **valid** variable names:

```
x
age
age_of_child
box1
box_2
_age
age_
```

The following are **invalid** variable names:

```
3
age of child
age-child
1st
2_box
```

What are the rules for naming variables?

Variable names conventions

Always choose variables with meaningful names:

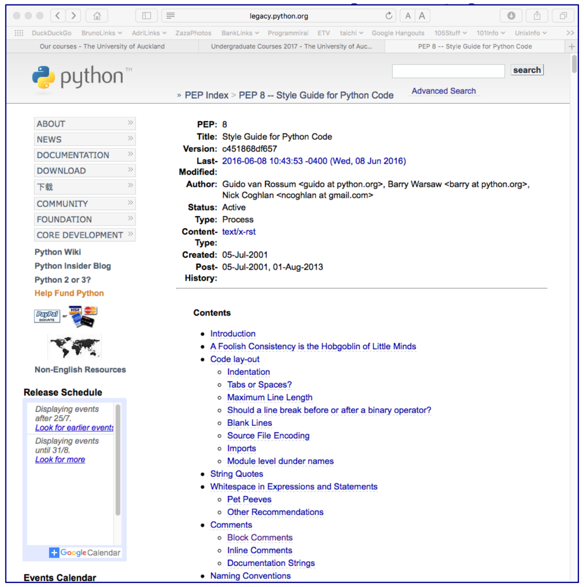
```
name
age
course
```

The convention when using multiple words, is to join words with an underscore:

```
user_input
age_allowed
age_of_child
circle_area
```

The convention is to use lower case letters for variable names.

Python is case sensitive. For example, the variable, `age`, is not the same as the variable, `Age`.



» PEP Index > PEP 8 -- Style Guide for Python Code

PEP:	8
Title:	Style Guide for Python Code
Version:	c451868df657
Last-Modified:	2016-06-08 10:43:53 -0400 (Wed, 08 Jun 2016)
Author:	Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>
Status:	Active
Type:	Process
Content-Type:	text/x-rst
Created:	05-Jul-2001
Post-History:	05-Jul-2001, 01-Aug-2013

Displaying events after 25/7.
Look for earlier events
Displaying events until 31/8.
Look for more

Contents

http://legacy.python.org/dev/peps/pep-0008/

Variable names should not be keywords

Variable names should not be keywords (also called **reserved words**):

```
and       elif     import    raise
as        else     in        return
assert    except   is        try
break     finally  lambda    while
class     for      nonlocal  with
continue  from     not       yield
def       global   or
del       if       pass
```

Look on page 11 of the reference book:

'**Think Python – How to think like a computer scientist**'.

The electronic copy of the reference book is available from:

https://www.cs.auckland.ac.nz/courses/compsci101s2c/resources/

What kind of information can our programs store?

Information in a program is categorised into different types. There are four basic types in Python:

- integer
- floating point
- string
- boolean

Integer values are numbers with no decimal point. They can be positive, 0 or negative:

```
202
0
-32
```

Floating point numbers are numbers with decimal points. They can be positive, 0 or negative:

```
1.0
-3.405
0.0
3.3333333
```

Note that the precision of floating point numbers is limited.

Assigning to a variable

The assignment operator **=** is used to assign a value to a variable, i.e., to store some information in the program memory:

```
result1 = 54
my_age = 21
bank_balance = 2019
```

The left hand side of the assignment operator is always a variable.

Doing Calculations

The following mathematical operators can be used with integers and with floating point numbers:

Addition	**+**
Subtraction	**-**
Multiplication	*****
Division	**/**
Exponentiation	******

```
result1 = 54 + 4
result2 = 15 / 5
bank_balance = 2019 * 2
solution = 4 ** 3
```

Expressions

An expression always evaluates to a single value, e.g.,

- **-3 + 5**
- **6 * 2 + 1**
- **52 – 3 * 2.3**
- **4.6**

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

```
result1 = 54 + 4 / 5
result2 = 15 / 5
bank_balance = 2019 * 3 / 100
age = 1 + 2 + 1 + 1
result3 = 7
```

Firstly the expression on the right hand side of the assignment operator is evaluated, and, then, the resulting value is assigned to the variable on the left hand side of the assignment operator.

Expressions

An expression can be used anywhere a single value can be used.

A variable can be used anywhere a single value can be used.

```
result1 = 54 + 4 / 5
result2 = result1 / 10
bank_balance = 2019 * 3 / result2

age = 5
age = age + 4
age = age * 3
```

Printing to the standard output

The **print() function** is used to print values to the standard output.

- `print(45.67)`
- `print(100000)`
- `print(44)`

Notice that round brackets (parentheses) are used with functions. After printing whatever is to be printed (whatever is inside the round brackets), a new line is printed, i.e., the pen moves to the next line.

```
principal = 100
years = 15

print(43)
print(principal)
print(2 * years + 1)
```

```
43
100
31
```

Printing a blank line

The **print()** statement with no arguments simply prints a blank line.

```
principal = 100
years = 15

print(43)
print()
print(principal)
print()
print()
print(2 * years + 1)
```

```
43

100


31
```

An Example Python Program

The formula for working out the final amount when a sum is invested at compound interest is: $M = P(1 + i)^n$ where:

- M is the final amount including the principal.
- P is the principal amount.
- i is the rate of interest (a whole number indicating the % interest) per year.
- n is the number of years invested.

Complete the code which calculates the final amount when $100 is invested for 15 years at 10% compound interest. The output prints the principal on one line followed by the final amount on the next line:

```
principal = 100
years = 15
rate = 10
final_amount =
```

```
100
417.7248169415656
```

Strings – Another Python Built-in Type

A string consists of a collection of characters delimited by single quotes (' … ') or by double quotes (" … "), e.g.,

- **"Meravigioso"**
- **'The final result is:'**
- **"5 + 2 * 5"**
- **""**

The program from the previous slide could be written with more information in the output:

```
principal = 100
years = 15
rate = 10
final_amount = principal * (1 + rate /100) ** years
print("Initial amount")
print(principal)
print("Final amount")
print(final_amount)
```

```
Initial amount
100
Final amount
417.7248169415656
```

Printing more than one value on a single line

The print() statement can be used to print more than one variable (or value) on a single line. Each value to be printed is separated by a comma, e.g.,

- print(1, "Meraviglioso", "Fabulous")
- print('The final results are:', 56, "and", 44)

```
1 Meraviglioso Fabulous
The final results are: 56 and 44
```

The default separator between the items to be printed is a single blank space, e.g.,

```
principal = 100
years = 15
rate = 10

final_amount = principal * (1 + rate /100) ** years
print("Initial amount", principal)
print("Final amount", final_amount)
```

```
Initial amount 100
Final amount 417.7248169415656
```

Printing more than one value on a single line

The default separator between the items printed is a single blank space. We can change this by including an optional last argument in the print() statement, **sep = "…"**, e.g.,

- print(1, "Meravigioso", "Fabulous", **sep = "*"**)
- print('The final results are:', 56, "and", 44, **sep = ""**)

```
1*Meravigioso*Fabulous
The final results are:56and44
```

The program from the previous slides can now be improved:

```
principal = 100
years = 15
rate = 10
final_amount = principal * (1 + rate / 100) ** years
print("Initial amount $", principal, sep = "")
print("Final amount $", final_amount, sep = "")
```

```
Initial amount $100
Final amount $417.7248169415656
```

Exercise

$1 NZ = $0.95 AUS. Write a program which converts $500 NZ to Australian dollars and converts $500 AUS to New Zealand dollars using the above exchange rate. The output of the program should be:

```
NZ $500 = AUS $475.0
AUS $500 = NZ $526.3157894736842
```

```
amount_to_convert = 500
nz_to_aus_rate = 0.95
nz_dollars = amount_to_convert
```

Summary

In a Python program we can:

- use variables, which have valid variable names, to store values,
- perform calculations using standard arithmetic operators
- describe differences between int and float types
- print numbers and strings to standard output

Examples of Python features used in this lecture

- use variables to store values, using valid variable names

```
principal = 100
years = 15
rate = 10
```

- perform calculations using standard arithmetic operators

```
final_amount = principal * (1 + rate / 100) ** years
```

- describe differences between int and float types

```
years = 15
rate = 0.01
```

- print numbers and strings to standard output

```
print("Final amount $", final_amount)
print()
print("Final amount $", final_amount, sep = "")
```