



# COMPSCI 101

## Principles of Programming

Lecture 2 – Variables, program execution,  
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 S1 C » Resources »

## Computer Science

### Resources: COMPSCI 101 Semester 1, 2020

- » Python
- » Reference Book
- » Miscellaneous

#### Python

- Python Download (Remember to get version 3, not version 2 - the latest is version 3.8.2)
- Python Documentation
- The Python Tutorial

<https://www.cs.auckland.ac.nz/courses/compsci101s1c/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

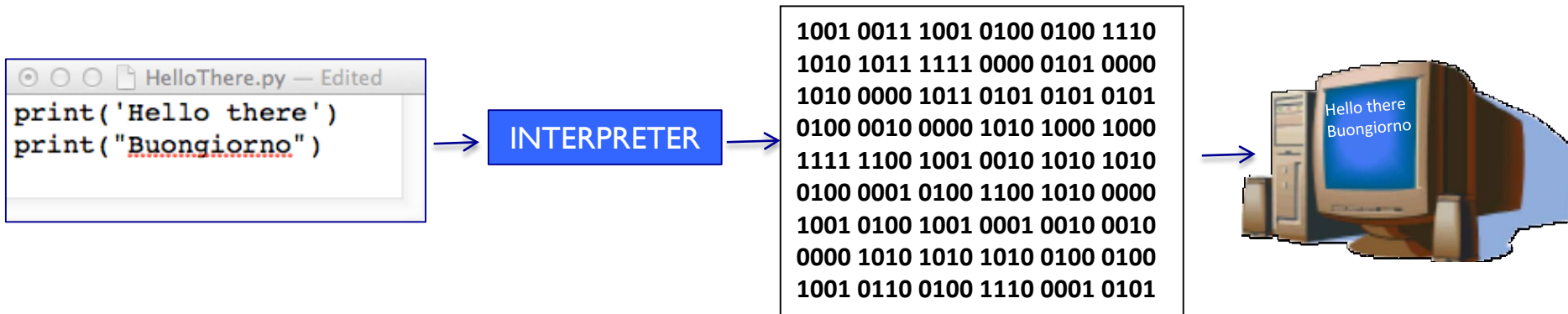


# The Python Interpreter

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.





# IDLE – The program editor used in COMPSCI 101

IDLE (**I**ntegrated **D**evelopment **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.

The screenshot shows two windows from the IDLE Python 3.6.4 environment. The top window, titled 'Greeting.py - C:/Users/dazh001/Desktop/Greeting.py (3.6.4)', contains the following Python code:

```
print("Hello Class")
print()
print("This is going to be fun!")
```

The bottom window, titled 'Python 3.6.4 Shell', shows the output of the script after execution. The shell displays the following text:

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1
900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more informati
on.
>>>
===== RESTART: C:/Users/dazh001/Desktop/Greeting.py
=====
Hello Class

This is going to be fun!
>>> |
```

The status bar at the bottom right of the shell window indicates 'Ln: 8 Col: 4'.



# Programs are Deterministic

---

## 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.
- ▶ then 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 = "Damir"
```

```
height = 183.0
```

```
age = 25
```





# 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 Naming 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`.



# Variable Names Should Not Be Key Words

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

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

Look on page 11 of the reference book:

- ▶ 'Think Python – How to think like a computer scientist'.
- ▶ <https://www.cs.auckland.ac.nz/courses/compsci101s1c/resources/>



# Style Guide

The screenshot shows a web browser window with the URL `legacy.python.org`. The page is titled "PEP 8 -- Style Guide for Python Code". The left sidebar contains navigation links: ABOUT, NEWS, DOCUMENTATION, DOWNLOAD, 下载, COMMUNITY, FOUNDATION, CORE DEVELOPMENT, Python Wiki, Python Insider Blog, Python 2 or 3?, Help Fund Python, Non-English Resources, Release Schedule, and Events Calendar. The main content area displays the following information:

- PEP:** 8
- Title:** Style Guide for Python Code
- Version:** c451868df657
- Last-Modified:** 2016-06-08 10:43:53 -0400 (Wed, 08 Jun 2016)
- Modified:**
- 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
- Type:**
- Created:** 05-Jul-2001
- Post-Create:** 05-Jul-2001, 01-Aug-2013
- History:**

The **Contents** section lists the following topics:

- Introduction
- A Foolish Consistency is the Hobgoblin of Little Minds
- Code lay-out
  - Indentation
  - Tabs or Spaces?
  - Maximum Line Length
  - Should a line break before or after a binary operator?
  - Blank Lines
  - Source File Encoding
  - Imports
  - Module level dunder names
- String Quotes
- Whitespace in Expressions and Statements
  - Pet Peeves
  - Other Recommendations
- Comments
  - Block Comments
  - Inline Comments
  - Documentation Strings
- Naming Conventions

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



# 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 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:  $M = P(I + i/100)^n$  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.

- ▶ "Great"
- ▶ '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, "Hello", "World")`
- ▶ `print('The final results are:', 56, "and", 44)`

```
1 Hello World  
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, "Hello", "World", sep = "*")`
- ▶ `print('The final results are:', 56, "and", 44, sep = "")`

```
1*Hello*World
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:

```
amount_to_convert = 500
nz_to_austr_rate = 0.95
nz_dollars = amount_to_convert
```

**NZ \$500 = AUS \$475.0**

**AUS \$500 = NZ \$526.3157894736842**





# 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 = "")
```