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

**Learning outcomes**

**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

**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**.

https://www.cs.auckland.ac.nz/courses/compsci101s1c/resources/
**IDLE – The program editor used in COMPSCI 101**

IDLE (Integrated Development Environment) 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.

**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.

**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.

```python
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`.

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/compsci101s1c/resources/](https://www.cs.auckland.ac.nz/courses/compsci101s1c/resources/)

Style guide

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

```python
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.

```python
principal = 100
years = 15
print(43)
print(principal)
print(2 * years + 1)
```

Printing a blank line

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

```python
principal = 100
years = 15
print(43)
print(principal)
```

An Example Python Program

The formula for working out the final amount when a sum is invested at compound interest is:

\[ M = P(1 + \frac{i}{100})^n \]

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:

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

```python
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)

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

```python
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 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 = ""`)

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_aus_rate = 0.95
AUS $500 = NZ $526.3157894736842
NZ $500 = AUS $475.0
```
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
  
  \[
  \text{principal} = 100 \\
  \text{years} = 15 \\
  \text{rate} = 10
  \]

- perform calculations using standard arithmetic operators
  
  \[
  \text{final\_amount} = \text{principal} \times (1 + \text{rate} / 100)^\text{years}
  \]

- describe differences between int and float types
  
  \[
  \text{years} = 15 \\
  \text{rate} = 0.01
  \]

- print numbers and strings to standard output
  
  \[
  \begin{align*}
  \text{print} ("\text{Final amount $}\), \text{final\_amount}) \\
  \text{print} () \\
  \text{print} ("\text{Final amount $}\), \text{final\_amount}, \text{sep} = "\)
  \end{align*}
  \]