

# COMPSCI 107

## Computer Science Fundamentals

Lecture 02 – Python syntax

variables

types

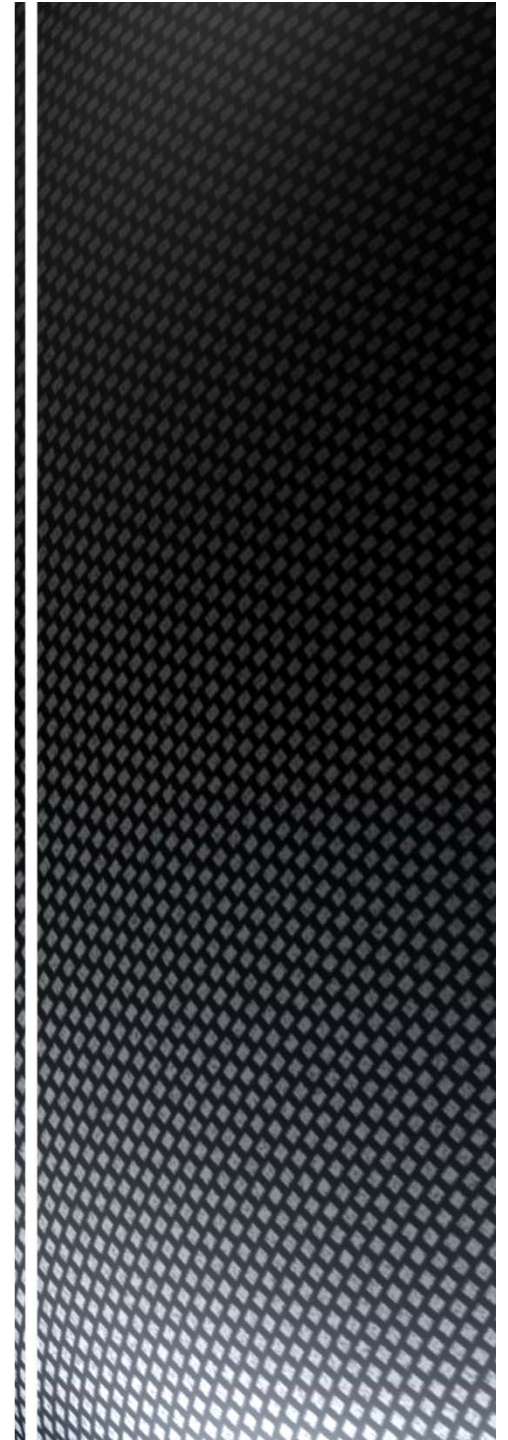
expressions

selection

iteration

functions

modules



# Learning outcomes

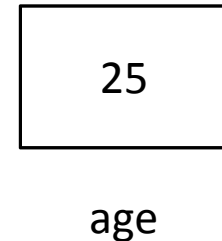
---

- At the end of this lecture, you should be able to:
  - Use variables to store values
  - Read from standard input and write to standard output
  - Convert variables from one type to another
  - Perform simple calculations using standard arithmetic operations
  - Use conditional statements to make decisions within a program
  - Use for and while loops to perform iteration
  - Define and call functions that accept arguments and return values
  - Use import statements and functions from the Python API
- Example code:
  - Convert a length from inches to centimetres
  - Calculate the area of a triangle
  - Print out the times tables
  - Determine if a given number is prime or not

# Reminder - Variables

---

- A simple storage box
  - name of the box is the identifier
  - stores only one thing at a time
- Information in a variable has a **type**
  - boolean, integer, float, string
- Values are stored in variables using an assignment statement
- Note: Python variables do not need to be declared before use



```
name = "Andrew"  
age = 25
```

# Tracing code

---

- Keep track of the contents of variables
  - Write down the name of each variable
  - Change the value when (and only when) an assignment occurs
  - When you change a value, cross out the old one and write a new one

length\_in\_inches: ~~50~~ 100

length\_in\_cms: 254.0

# Exercise

- What values are stored in a, b and temp after executing the following code? Perform a code trace.

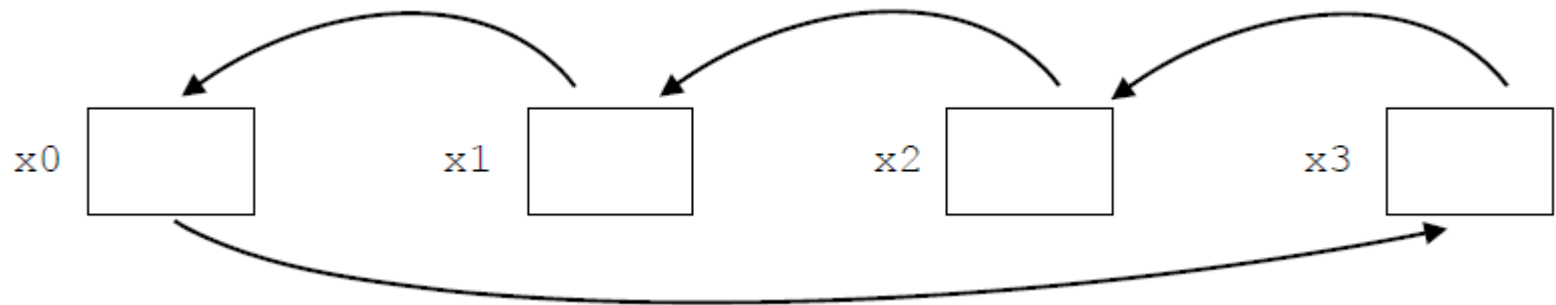
```
a = 7
b = 3

temp = a
a = b
b = temp
```

- Explain the purpose of the code shown above. What does it do?

# Exercise

- Suppose that there are 4 variables names `x0`, `x1`, `x2` and `x3`. Write the code to move the values stored in those variables to the left, with the leftmost value ending up in the rightmost variable, as shown in the diagram below. You may use a temporary variable.



# Python input and output

---

- Python provides simple functions for reading and writing text
- `input([arg])` function
  - returns a string containing text entered from keyboard
  - removes the newline terminator at the end of the string
  - optional argument is used as a prompt
- `print([args])` function
  - prints the argument to the standard output (screen)
  - multiple arguments will be printed with a space separating each argument

```
name = input('Enter your name: ')  
print('Hello', name)
```

# Exercise

- Why is the idea of a type useful to us? In other words, why do languages have the notion of type?
- What is the advantage of static type checking compared with dynamic type checking?



# Data Types

---

- All data is ultimately stored as numbers
  - We can interpret the numbers in different ways
  - Type determines how to interpret the numbers
  - Type determines which operations are permitted
- Static type checking
  - Type errors are detected at compile time
  - Automatically checked
- Dynamic type checking
  - Type errors are detected at run time
  - Manually debugged

# Types

- Python variables are dynamically typed

- a variable can hold any type of value

```
x = 3  
print(x)
```

```
x = 'Hello'  
print(x)
```

```
3  
Hello
```

- Checking the type

- type() function

```
x = 3  
print(type(x))
```

```
x = 'Hello'  
print(type(x))
```

```
<class 'int'>  
<class 'str'>
```

# Type casting

---

- Changing the way data is interpreted (changing the type)
  - Use the name of the type as a function
  - `int( x )`
  - `float( x )`
  - `str( x )`

```
x = input('Enter a number: ')\nprint(type(x))
```

```
y = int(x)\nprint(type(y))
```

```
z = float(x)\nprint(type(z))
```

```
Enter a number: 56\n<class 'str'>\n<class 'int'>\n<class 'float'>
```

# Exercise

- Write code that reads an integer value from standard input, and prints that value to standard output.
- Note: The function to read from standard input is:
  - `input([prompt])`
- Note: The `input()` function returns a string

# Arithmetic operations

---

- Mathematical operators

- Addition +
- Subtraction -
- Multiplication \*
- Division /
- Exponentiation \*\*
- Floor division //
- Modulus %

- “In place” operators

- $x += 1$
- $x -= 1$
- $x *= 2$
- ...

is the same as

$x = x + 1$

is the same as

$x = x - 1$

is the same as


$x = x * 2$

# Expression

---

- An expression is part of the program that can be evaluated (i.e. when the computer follows the rules that define the instructions, an expression turns into a value).

`x = 3 + 4`

 3 + 4 is an expression

- An expression can be used anywhere that a value is used

# Example

---

- Convert a length from inches to centimetres

```
length_in_inches = 100  
  
length_in_cm = length_in_inches * 2.54  
  
print(length_in_cm)
```

- Floor division and modulus
  - Integer part of a division, and remainder after division
- What do each of the following expressions evaluate to?
  - $10 + 4$
  - $10 - 4$
  - $10 * 4$
  - $10 / 4$
  - $10 ** 4$
  - $10 // 4$
  - $10 \% 4$



# Boolean values and related operators

---

- Boolean values
  - True
  - False
- Relational operators
  - `>`, `>=`, `<`, `<=`, `==`
- Boolean operators
  - `and`, `or`, `not`

```
>>> 2 == 3
False
>>> 2 == 3 or 4 < 5
True
```

# Conditionals

---

- Code is executed if the condition is true

```
if name == "Andrew":  
    print("Hi Andrew")
```

```
if n % 2 == 0:  
    print("Even number")  
else:  
    print("Odd number")
```

```
if n < 0:  
    print("Negative number")  
elif n > 0:  
    print("Positive number")  
else:  
    print("Zero")
```

- For loops

- Used to iterate through a sequence of values

```
for count in range(0, 100):  
    print(count)
```

- While loops

- Used to repeat statements when the end condition is unknown

```
goal = 'catfish'  
password = ''  
while password != goal:  
    password = input('Enter the password: ')  
print('Correct')
```

# Range

---

- Used to generate integer numbers within a range of values

- Includes the start value, but excludes the stop value

- `range(stop)`

- range of values from 0 to stop value

```
range(10)
```

```
0 1 2 3 ... 8 9
```

- `range(start, stop)`

- range of values from start to stop value

```
range(3, 8)
```

```
3 4 5 6 7
```

- `range(start, stop, step)`

- range of values from start to stop value, increasing by step each time

```
range(4, 10, 2)
```

```
4 6 8
```

```
range(10, 4, -2)
```

```
10 8 6
```

# Functions

---

- A function is a sequence of instructions designed to perform a task, and is packaged as a unit.
  - Functions have a name
  - Functions accept arguments
  - Functions return values

```
def rectangle_area(width, height):  
    return width * height
```

- Syntax
  - Indentation rather than braces are used to signify blocks of code
  - Variables defined within the *scope* of a function are not available outside the function

# Exercises

- Write a function called `triangle_area` that calculates the area of a triangle given the base and height values

- $\text{area} = \frac{1}{2} \text{base} * \text{height}$

```
>>> a = triangle_area(10, 20)
>>> print(a)
100.0
```

- Write a function that asks the user for the times tables to print, then prints out the times tables from 1 to 10 (inclusive)

```
Enter the times tables: 4
4 x 1 = 4
4 x 2 = 8
...
4 x 10 = 40
```

# Example: Prime numbers

---

- Determine if a number is prime or not
  - A number less than 2 is not prime
  - A number equal or greater than 2 is prime if it is not divisible by any other number except itself and 1

```
def is_prime(n):  
    if n < 2:  
        return False  
    for i in range(2, n):  
        if n % i == 0:  
            return False  
    return True
```

# Importing modules

---

- Code is stored in modules
  - We want to reuse code as much as possible
  - Build up libraries of code
  - Importing a module allows you to access code in that module

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.sqrt(4)
2.0
```

- Python.org has a list of all the modules and functions provided



# Built-in functions

---

- Python provides a wide range of built-in functions, including
  - `round(value [, number of decimal places])`
  - `max(value, value [, value ...])`
  - `min(value, value [, value ...])`
  - `float(value)`
  - `int(value)`
  - `str(value)`
  - `type(value)`

# Summary

---

- Variables in Python are dynamically typed and do not need to be declared before using them
- Conditional statements have optional else and elif clauses
- For loops are used to iterate through a sequence of values
- While loops are used when the end condition is unknown
- Code can be more easily reused when it is placed in a function or module
- Modules may be imported to access the functions stored in them
- Variables (including parameters) defined within a function may not be accessed outside the function.