Programming Basics

15-110 – August 08/28

Announcements

- Moved from Ed to Piazza

- 65% of students completed Ex1-1
  - If you haven't completed it yet, you still can! Exercises can be submitted late under the revision policy.

- 80% of students completed Pre-Semester Survey
  - Due tonight at 11:50pm

- Check1 due next Tuesday at noon
  - Tutorial: how to download & work on written assignments
  - Tutorial: how to submit files on Gradescope

Learning Objectives

- Recognize and use the basic **data types** in programs
- Interpret and react to basic **error messages** caused by programs
- Use **variables** in code and trace the different values they hold

Python & IDEs

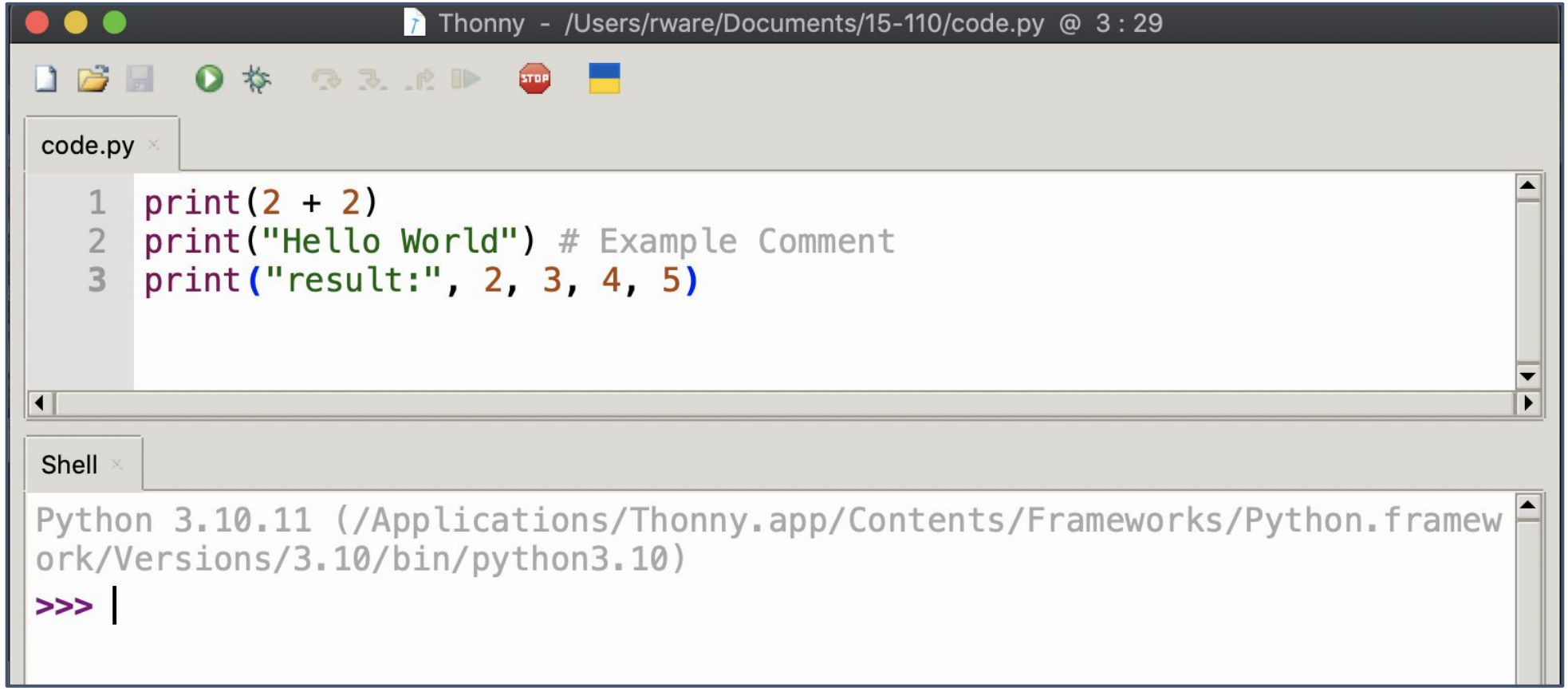
Algorithms can be expressed as **programs** in many different **programming languages**.

Python programming language **syntax:**

code.py

```
1  print(2 + 2)
2  print("Hello World") # Example Comment
3  print("result:", 2, 3, 4, 5)
4
```

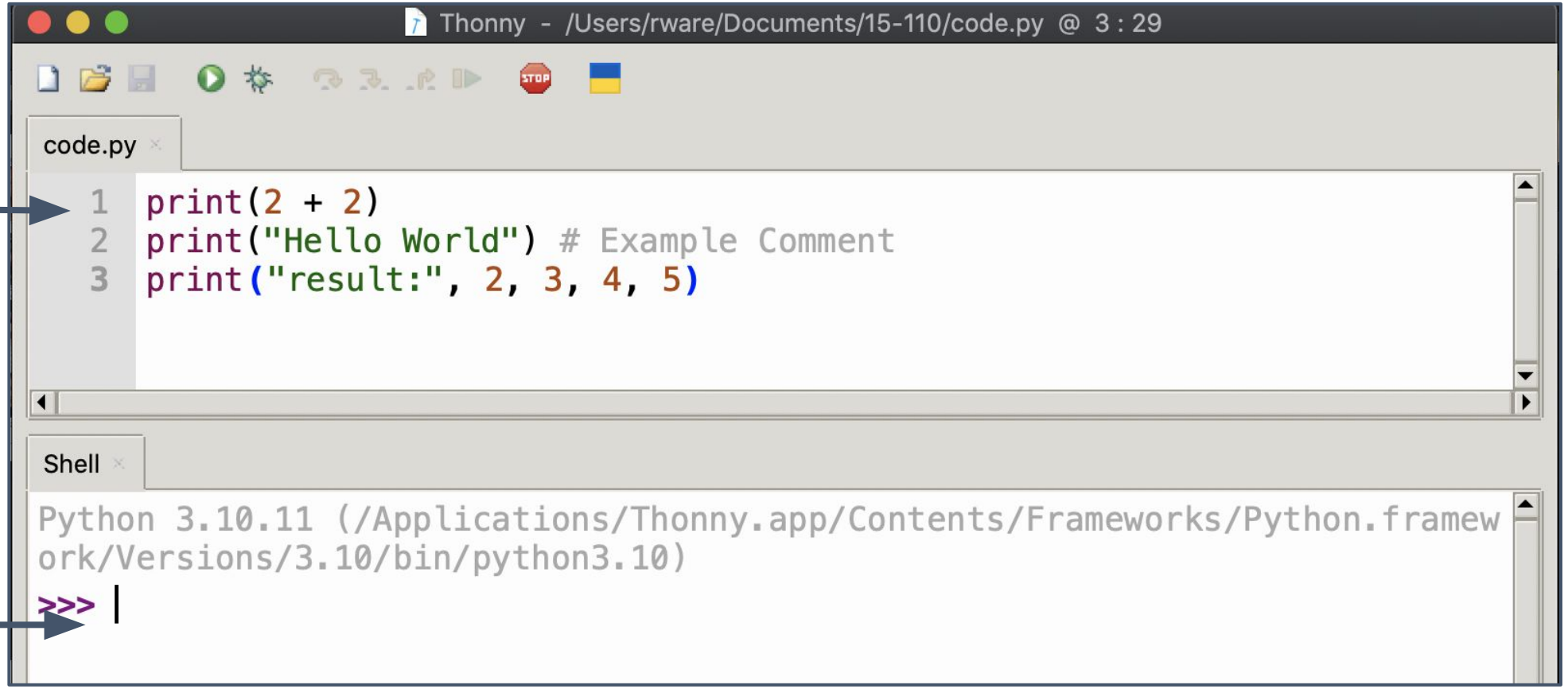
An **IDE** (Integrated Development Environment) is a text editor for programs.



Thonny is an IDE with two parts for writing code:
a **text editor** and an **interpreter (shell)**.

text editor

interpreter



Data Types

Data is information we can manipulate with **operations**.

Data have different **types** based on properties:

Numbers:

```
10
-412
1.0
0.333
1e10
```

Strings:

```
“Hello world”
“Dear Prof”
“”
“    be kind    ”
```

Truth values (booleans)

```
True
False
```

Numbers have two types: integers and floating point.

Integers are whole numbers, positive or negative: `0` `14` `-7`

Floating point numbers include a decimal point: `3.0` `5.735` `8e10`

Numbers can be combined with math **operators**:

addition `+`	subtraction `-`	multiplication `*`	division `/`	power `**`

to make **expressions**:

```
>> 4**2+(5-2)/3
17.0
```

Text values are called **strings**.

Text is recognized by Python as a string by putting it into either **single quotes**: `'Hello'` or **double quotes**: `"Hello"`

Strings can be **concatenated** using addition operator `+`:

```
>> "Hello" + "World"
HelloWorld
```

Strings can be **repeated** using multiplication operator `*`:

```
>> "Hello" * 3
HelloHelloHello
```

Python can evaluate whether certain expressions are true or false. These types of values are called **Booleans**.

Booleans are either `True` or `False`

We get a Boolean when we do a comparison with **comparison operators**:

less than `<`	greater than `>`	equal `==`
less or equal `<=`	greater or equal `>=`	not equal `!=`

```
>> “Hello” == “World”
False
```

Mixing types in Python can cause **error messages** when the types do not go together well.

Adding strings and numbers results in a **TypeError**.

```
>> “Hello” + 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Comparing strings and numbers results in a **TypeError**.

```
>> “Hello” < 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'str' and 'int'
```

Mixing types in Python can cause **error messages** when the types do not go together well.

Integers and floating point numbers **can be mixed** freely. Usually the result is a floating point number.

```
>> 8 * 2.0
16.0
```

Python uses **shortened names** for data types in error messages:

Integers are called `int`

Floating point numbers are called `float`

Strings are called `str`

Booleans are called `bool`

Activity: Predict the Type

Let's do a poll to see if you can identify data types correctly! For each expression, vote for the type you think it will evaluate to!

Hold up 1, 2, 3, or 4 fingers to indicate your vote:

- 1: bool
- 2: float
- 3: int
- 4: string

```
"1" + "2"
```

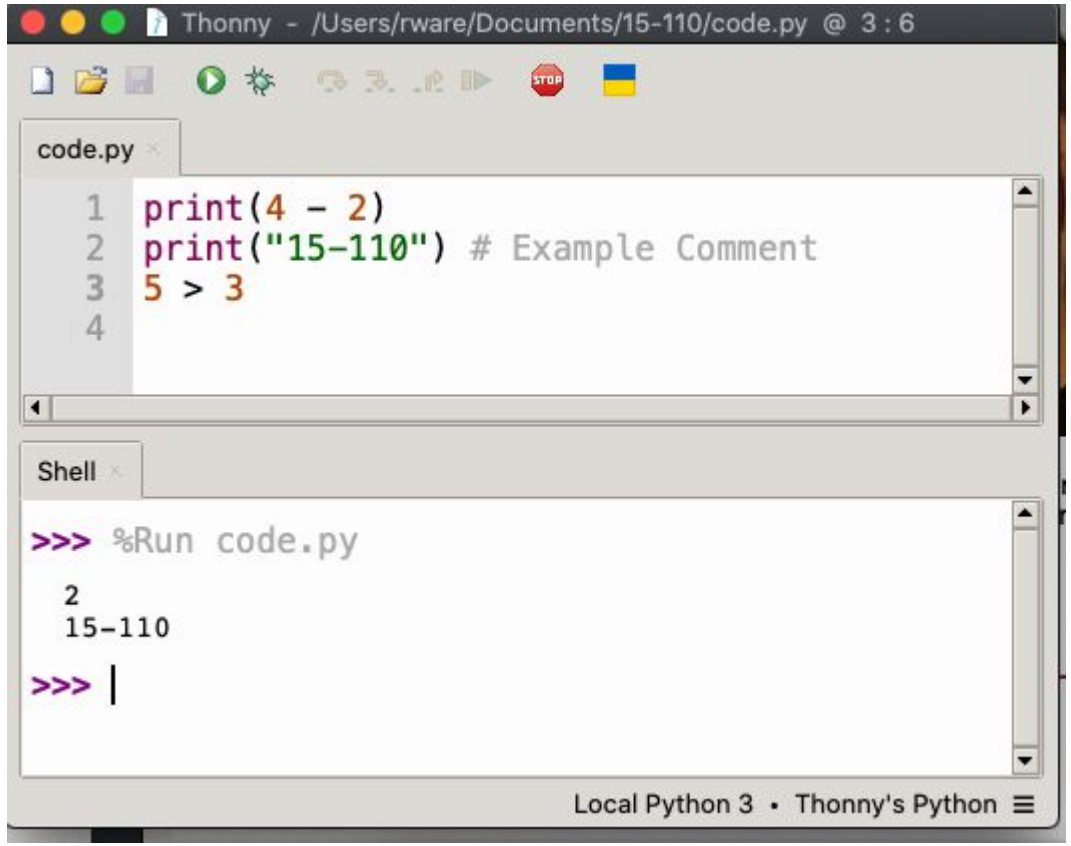
```
15-110
```

```
"Hello" == "World"
```

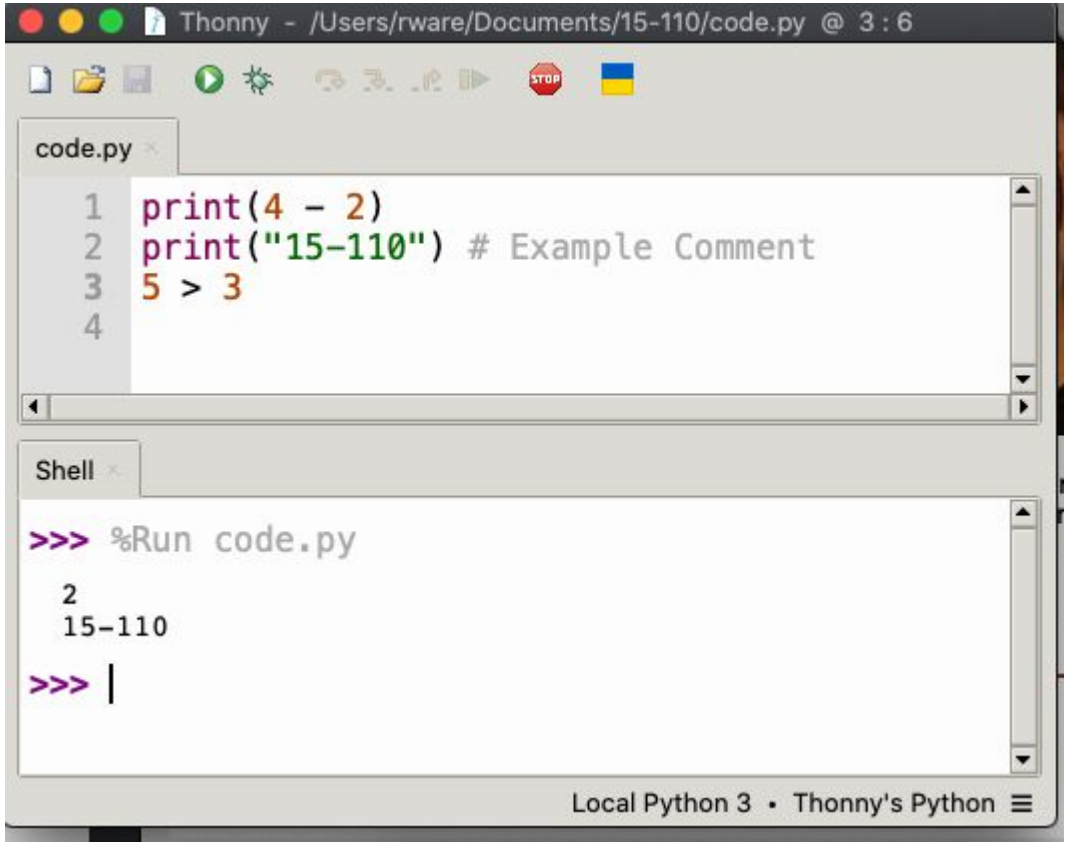
```
3.0 * 5.0
```

Writing Code in Files

To write **longer programs**, write lines of code in **editor**, save 💾 and then run ▶️



`print` **displays** data in interpreter



If you want to display **multiple values** in the interpreter on the same line, you have two choices.

If printing strings can concatenate together:

```
>> print(“Result: ” + “2”)
Result: 2
```

Use commas to separate values (will automatically print with spaces between values):

```
>> print(“Result:”, 2)
Result: 2
```

Comments are notes for humans and are ignored by the computer.

Any text that follows a # will be ignored by the computer:

```
print(“Hello World”) # a greeting
```

To comment out a block of code, use triple quotes at beginning and end:

```
““““
print(“ignore”)
print(“this”)
””””
```

Error Messages

Errors happen when syntax is not valid Python code.

```
>> Print(“Hello World”)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined

>> print “Hello World”
  File "<stdin>", line 1
    print "Hello World"
    ^^^^^^^^^^^^^^^^^^^
SyntaxError: Missing parentheses in call to
'print'. Did you mean print(...)?
```

Debug errors by reading the message.

1. Look for the line number. This line tells you approximately where the error occurred.
2. Look at the error type.
3. If it says SyntaxError, look for the inline arrow. The position gives you more information about the location of the problem (though it isn't always right).
4. If it says something else, read the error message. The error type and its message give you information about what went wrong.

We'll talk more about the debugging process in future lectures.

1-2.py

```
1 print(Hello World)
2 Print("Hello World")
```

```
>>> %Run test.py
 Traceback (most recent call last):
   File "C:\Users\river\Downloads\test.py", line 1
     print(Hello World)
                     ^
 SyntaxError: invalid syntax

>>>
```

inline arrow

1-2.py

```
1 print("Hello World")
2 Print("Hello World")
```

```
>>> %Run test.py
 Hello World
 Traceback (most recent call last):
   File "C:\Users\river\Downloads\test.py", line 2, in <module>
     Print("Hello World")
 NameError: name 'Print' is not defined

>>>
```

error type

Be careful when using **whitespace** (spaces, tabs, and the return key) – it can sometimes count as syntax too!

```
      print("Hello World") # IndentationError

p r i n t ( "Hello World" ) # SyntaxError

print ( "Hello World" ) # this is okay!
```

Variables

Variables let us store data so we can reuse it in future computations.

We define a variable with an **equal sign**:

```
variable = value
```

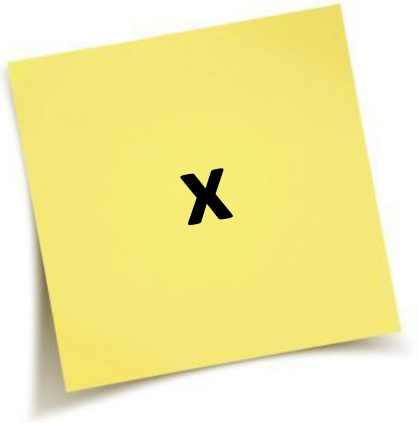
Variables can only go on the left side of this code, and its value (or an expression that evaluates to a value) goes on the right. For example:

```
myPet = "Kimchee"
result = 5 + 2    # result is set to 7
42 = foo # SyntaxError
```

Variables are like **sticky notes**.

You can think of a variable as a sticky note that is applied to a data value.

When you want to use the data value, you can use it directly or refer to the name on the note.



You assign a variable to a value by writing the name on the note and putting the note on the value.

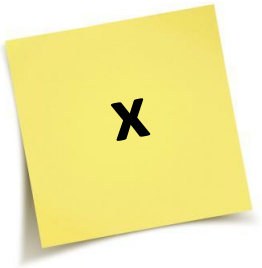
We can use variables in **expressions** and we can change the variable to hold a new value.

```
x = 5
y = x - 2  # x evaluates to 5
```

By changing the values of variables we are updating the **program state**:

```
x = 5
x = x - 1 # x evaluates to 5 on the right
          # then changes to 4
print("x:", x) # x: 4
```

This is like moving the sticky note to a new value



Python runs **every line in order** and doesn't peek ahead

If you want to use a variable, you must define it before it is used!

```
print(foo) # this causes an error!
foo = 42
```

```
foo = 42
print(foo) # this is fine!
```

Activity: Trace the Variable Values

You do: Trace through the following lines of code. What values do a and b hold at the end?

```
a = 4
b = 7
b = a - 2
a = a + 1
```

Sidebar: Rules for Variable Names

Variable names can use any combination of uppercase letters, lowercase letters, digits, and underscores. They must start with a letter or _. Starting with a lowercase letter is recommended.

Variable names are case sensitive. For example, Banana is not the same as banana. Make sure to type your variables correctly, or you'll get a NameError!

Learning Objectives

- Recognize and use the basic **data types** in programs
- Interpret and react to basic **error messages** caused by programs
- Use **variables** in code and trace the different values they hold