CFS2160: Software Design and Development

02: Representing Things

Objects. Variables. Expressions.

Tony Jenkins
A.Jenkins@hud.ac.uk

Recap: Jobs

Before next week, make sure you have:

- ➢ Got the book, and read the first two chapters.
- ➢ Started to work with PyCharm, using the Python console.
- ➢ Completed a Git tutorial.

And, optionally:

- ➢ Connected PyCharm to your GitHub account.
- ➢ Understood "Public" and "Private" Git repositories.

Recap: Jobs

Before next week, make sure you have:

- ➢ Got the book, and read the first two chapters.
- ➢ Started to work with PyCharm, using the Python console.
- ➢ Completed a Git tutorial.

And, optionally:

- ➢ Connected PyCharm to your GitHub ac
- ➢ Understood "Public" and "Private" Git

As before, there is a lot here.

If you are in doubt, or can't work something out - Ask!

Recap: Jobs

Before next week, make sure you have:

- ➢ Got the book, and read the first two chapters.
- ➢ Started to work with PyCharm, using the Python console.
- ➢ Completed a Git tutorial.

And, optionally:

- ➢ Connected PyCharm to your GitHub ac
- ➢ Understood "Public" and "Private" Git

From now on, 10 minutes of every lecture are kept for answering (anonymous) questions!

If in doubt - Ask!

Data Processing

Any computer program processes data.

Usually, this data represents "things" in the "real world".

So, a programmer's job consists of:

- ➢ Devising a way to represent real world things in a program.
- ➢ Writing code to manipulate this representation.
- ➢ Storing and/or displaying the results.

Representation

Some things are easy to represent with what we know so far:

- The number of eggs in a box is an integer.
- A temperature reading is a floating-point number.
- A message is a string.

Representation

Some things are easy to represent with what we know so far:

- The number of eggs in a box is an integer.
- A temperature reading is a floating-point number.
- A message is a string.

Some more complex things are harder to represent:

- A person.
- A group of people (maybe a football team).
- A set of weather readings taken together at the same time.

Variables

Last week, we saw that a *variable* is created by assigning it a value:

- `eggs = 13`
- `temperature = 16.5`

Variables

Last week, we saw that a *variable* is created by assigning it a value:

- `eggs = 13`
- `temperature = 16.5`

In addition we noted that a variable has a *type*.

- `eggs` is an integer.
- `temperature` is a floating-point number.

Variables

Last week, we saw that a *variable* is created by assigning it a value:

- `eggs = 13`
- `temperature = 16.5`

In addition we noted that a variable has a *type*.

- `eggs` is an integer.
- `temperature` is a floating-point number.

Finally, we saw that every data type has a set of operations that manipulate its values.

Objects and Classes

Keen-eyed viewers will have noticed that when Python is asked what type a value is, it used the word "class":

```
>>> type (1)
<class 'int'>
```

In Python, everything (i.e. every variable) is an *Object*.

Objects of the same type are grouped into *Classes*.

A class can represent a simple number, or something more complex, like a person.

Objects and Classes

Keen-eyed viewers will have noticed that when Python is asked what type a value is, it used the word "class":

```
>>> type (1)
<class 'int'>
```

In Python, everything (i.e. every variable) is an *O*

Objects of the same type are grouped into *Classe*

A class can represent a simple number, or somet

This terminology will return when we move to Java.

In Python it's a bit more "behind the scenes".

Identifiers

An object has a name, correctly called an *identifier.*

A good identifier includes information about the purpose of the object.

For example:

- ```
  e = 13
  ```
  ```

Identifiers

An object has a name, correctly called an *identifier.*

A good identifier includes information about the purpose of the object.

For example:

- ~~`e = 13`~~
- `eggs = 13`

Identifiers

An object has a name, correctly called an *identifier.*

A good identifier includes information about the purpose of the object.

For example:

- ~~`e = 13`~~
- `eggs = 13`
- `the_number_of_eggs_in_the_basket = 13`

Identifiers

An object has a name, correctly called an *identifier.*

A good identifier includes information about the purpose of the object.

For example:

- ~~`e = 13`~~
- `eggs = 13`
- ~~`the_number_of_eggs_in_the_basket = 13`~~

Identifiers

An object has a name, correctly called an *identifier.*

A good identifier includes information about the purpose of the object.

Identifier names follow conventions:

- ➢ Start with a lowercase letter.
- ➢ Are written in a consistent form:
  - ➢ `thisIsCamelCase`
  - ➢ `this_is_snake_case`

Identifiers

An object has a name, correctly called an *identifier.*

A good identifier includes information about the purpose of the object.

Identifier names follow conventions:

- ➢ Start with a lowercase letter.
- ➢ Are written in a consistent form:
  - ➢ `thisIsCamelCase`
  - ➢ `this_is_snake_case`

Camel Case is usual in Java.

Snake Case is usual in Python.

Identifier Rules

Every language defines rules that determine what is valid as an identifier.

In Python, an identifier:

- ➢ Must start with a letter (or an underscore, but that has a special meaning).
- ➢ Can include letters, numbers, or underscores.
- ➢ Is case-sensitive.
- ➢ Can be any length.

In addition, every language has "reserved words" that have a special meaning, and cannot therefore be used as identifiers.

Naming Conventions

In addition, by *convention*, certain things in programs are named in particular ways.

This improves readability, and helps other programmers understand code.

Failing to follow conventions will confuse others.

By convention:

- `knights` is a variable.
- `Knights` is a class.
- `number_of_knights` is a variable, but `NUMBER_OF_KNIGHTS` is a constant.

Naming Conventions

In addition, by *convention*, certain things in programs are named in particular ways.

This improves readability, and helps other programmers understand code.

Failing to follow conventions will confuse others.

By convention:

- `knights` is a variable.
- `Knights` is a class.
- `number_of_knights` is a variable, l
  constant.

Python conventions for this (and much else) are in PEP 8.

https://www.python.org/dev/peps/pep-0008/

Reserved Words

A common "gotcha" (mistake for newcomers to a language) is to use a reserved word for an identifier.

Ways to avoid this are:

- Get a list of such words.
- Use an IDE with syntax highlighting.

Reserved Words

A common "gotcha" (mistake for newcomers to a language) is to use a reserved word for an identifier.

Ways to avoid this are:

- ➢ Get a list of such words.
- ➢ Use an IDE with syntax highlighting.

What happens if an error is made can depend on language, and precisely what sort of reserved word is used.

```
>>> for = 3
  File "<stdin>", line 1
    for = 3
        ^
SyntaxError: invalid syntax

>>> print = 3
>>> print
3
>>> print ('Hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
TypeError: 'int' object is not
callable
```

Reserved Words

A common "gotcha" (mistake for newcomers to a language) is to use a reserved word for an identifier.

Ways to avoid this are:

If you get an odd error like these, a good thing to check first is whether the identifier is a reserved word.

```
>>> for = 3
  File "<stdin>", line 1
    for = 3
        ^
SyntaxError: invalid syntax

>>> print = 3
>>> print
3
>>> print ('Hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in
<module>
TypeError: 'int' object is not
callable
```

Other Types

There are other types available in Python.

```
>>> readings = [18, 12, 27, 8]
```

Other Types

What's going on here?

```
>>> readings = [18, 12, 27, 8]
```

Other Types

What's going on here?

```
>>> readings = [18, 12, 27, 8]

>>> max (readings)
27
```

Other Types

What's going on here?

```
>>> readings = [18, 12, 27, 8]

>>> max (readings)
27

>>> readings.sort ()
>>> readings
[8, 12, 18, 27]
```

Other Types

What's going on here?

```
>>> readings = [18, 12, 27, 8]

>>> max (readings)
27

>>> readings.sort ()
>>> readings
[8, 12, 18, 27]

>>> sum (readings) / len (readings)
16.25
```

Other Types

The variable with identifier `readings` is a *list*.

A list contains a collection of values (objects), usually of the same type.

```
>>> readings = [18, 12, 27, 8]

>>> max (readings)
27

>>> readings.sort ()
>>> readings
[8, 12, 18, 27]

>>> sum (readings) / len (readings)
16.25
```

Other Types

The variable with identifier `readings` is a *list*.

A list contains a collection of values (objects), usually of the same type.

So `readings` is a list of integers.

Which means there is a set of operations that will do various useful things with the list.

```
>>> readings = [18, 12, 27, 8]

>>> max (readings)
27

>>> readings.sort ()
>>> readings
[8, 12, 18, 27]

>>> sum (readings) / len (readings)
16.25
```

Other Types

The variable with identifier `readings` is a *list*.

A list contains a collection of values (objects), usually of the same type.

So `readings` is a list of integers.

Which means there is a set of operations that will do various useful things with the list.

(Technically, `readings` is a *composite* or *compound* data type.)

```
>>> readings = [18, 12, 27, 8]

>>> max (readings)
27

>>> readings.sort ()
>>> readings
[8, 12, 18, 27]

>>> sum (readings) / len (readings)
16.25
```

Primitive Types

Most languages define a collection of basic types, usually called *primitive* types.

Python, as we have seen, defines four:

- Integers.
- Floating-point numbers.
- Booleans.
- Strings.

Primitive Types

Most languages define a collection of basic types, usually called *primitive* types.

Python, as we have seen, defines four:

- Integers.
- Floating-point numbers.
- Booleans.
- Strings.

(And one more, to represent not having a value.)

Primitive Types: Integers ('int')

Most languages define a collection of basic types, usually called *primitive* types.

Python, as we have seen, defines four:

- ➢ **Integers.**
- ➢ Floating-point numbers.
- ➢ Booleans.
- ➢ Strings.

(And one more, to represent not having a value.)

An Integer is a whole number.

Operations include addition, division, modulus, etc.

Integers can be negative.

Usage:

```
>>> knights = 6
>>> brave_knights = knights - 1

>>> knights += 1
>>> knights -= 1
```

Primitive Types: Floating-Point ('float')

Most languages define a collection of basic types, usually called *primitive* types.

Python, as we have seen, defines four:

- ➢ Integers.
- ➢ **Floating-point numbers.**
- ➢ Booleans.
- ➢ Strings.

(And one more, to represent not having a value.)

A Floating-point number has a decimal part.

Operations include addition, division, etc.

Floating-point numbers can be negative.

Usage:

```
>>> swallow_speed = 12.2
```

```
>>> swallow_speed *= 0.5
>>> swallow_speed /= 2.0
```

Primitive Types: Booleans ('bool')

Most languages define a collection of basic types, usually called *primitive* types.

Python, as we have seen, defines four:

- Integers.
- Floating-point numbers.
- **Booleans.**
- Strings.

(And one more, to represent not having a value.)

A Boolean value is either True, or False.

Operations include AND, OR, NOT.

Usage:

```
>>> brave_knight = False
>>> brave_knight = not True

>>> bold_knight = True
>>> brave_knight and bold_knight
False
```

Primitive Types: Strings ('str')

Most languages define a collection of basic types, usually called *primitive* types.

Python, as we have seen, defines four:

- Integers.
- Floating-point numbers.
- Booleans.
- **Strings.**

(And one more, to represent not having a value.)

A string is any sequence of characters.

Operations include addition, multiplication.

Plus many string-specific operations.

Usage:

```
>>> name = "Sir Robin"
>>> full_name = "Brave " + name

>>> name.find ('r')
2
```

Primitive Types: None ('NoneType')

Most languages define a collection of basic types, usually called *primitive* types.

Python, as we have seen, defines four:

- ➢ Integers.
- ➢ Floating-point numbers.
- ➢ Booleans.
- ➢ Strings.

(And one more, to represent not having a value.)

This is the case where a variable has no value.

The variable does *exist* but it contains no value.

The value it does contain is None.

Usage:

```
>>> nothing = None

>>> type (nothing)
<class 'NoneType'>
>>> not nothing
True
```

Complete Programs

We now have almost enough to write a complete program.

We can:

- ➢ Store values in appropriate variables.
- ➢ Manipulate the values.
- ➢ Display the results.
  - ➢ (The `print` statement.)

One thing remains: how to get some input from our user?

Complete Programs

We now have almost enough to write a complete program.

We can:

- ➢ Store values in appropriate variables.
- ➢ Manipulate the values.
- ➢ Display the results.
  - ➢ (The `print` ***statement***.)

One thing remains: how to get some input from

A program is a collection of *statements*.
We have statements to manipulate values, and to display results.
We need one more …

Need Input

The `input` statement takes a value from the keyboard, and assigns it to a variable.

Optionally, it also displays a message to the user:

- ➢ `name = input ('Good Morning. What is your name? ')`
- ➢ `eggs = input ('How many eggs have you got? ')`

Need Input

The `input` statement takes a value from the keyboard, and assigns it to a variable.

Optionally, it also displays a message to the user:

```
➢ name = input ('Good Morning. What is your name? ')
➢ eggs = input ('How many eggs have you got? ')
```

The value is *always* read as a string, so if a number is needed it must be converted:

```
➢ eggs = int (input ('How many eggs have you got? '))
```

Take a close look at the brackets!

Need Input

The `input` statement takes a value from the key

Note that if the user enters something that is not an integer, a *run-time error* will happen.

We'll see how to handle those later.

Optionally, it also displays a message to the user

➢ `name = input ('Good Morning`

➢ `eggs = input ('How many egg`

The value is *always* read as a string, so if a number is needed it must be converted:

➢ `eggs = int (input ('How many eggs have you got? '))`

Take a close look at the brackets!

A Complete Program

"Our students take five exams.

The results are all whole numbers, between 0 and 100.

The final mark they get is the average of the five results.

We need a program to read the student's name, and their five results, then calculate the average, and finally display the final mark."

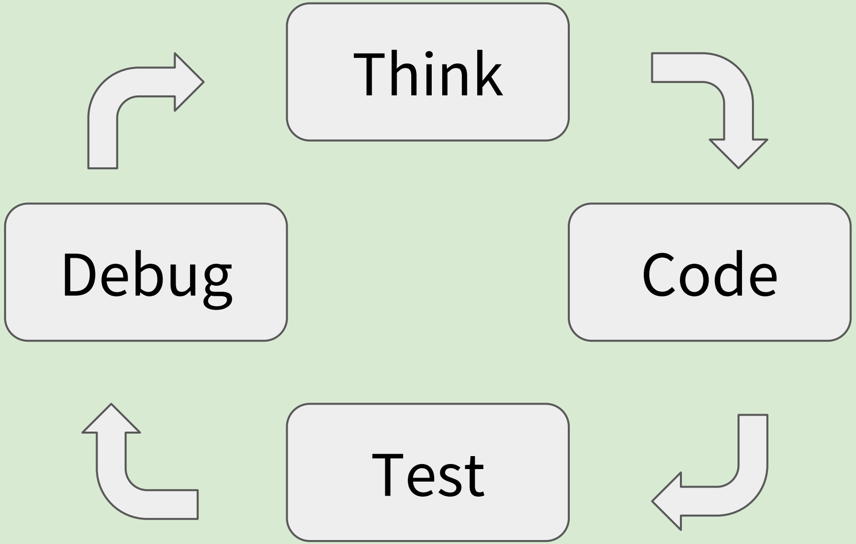
A Complete Program

"Our students take five exams.

The results are all whole numbers, between 0 and 100.

The final mark they get is the average of the five results.

We need a program to read the student's name, and their five results, then calculate the average, and finally display the final mark."



A Complete Program

"Our students take five exams.

The results are all whole numbers, between 0 and 100.

The final mark they get is the average of the five results.

We need a program to read the student's name, and their five results, then calculate the average, and finally display the final mark."

We can break the problem down to something like:

1. Read the student's name.
2. Read the five results, one at a time.
3. Find the total of the results, and divide by five to get the average.
4. Display the student's name along with their average.

A Complete Program

"Our students take five exams.

The results are all whole numbers, between 0 and 100.

The final mark they get is the average of the five results.

We need a program to read the student's name, and their five results, then calculate the average, and finally display the final mark."

We can break the problem down to something like:

1. Read the student's name.
2. Read the five results, one at a time.
3. Find the total of the results, and divide by five

Important

We are writing this program with the Python we have met so far. So the result is really not the best way to do it!

```
name = input ('Enter the student\'s name: ')

mark_1 = int (input ('Enter first result:  '))
mark_2 = int (input ('Enter second result: '))
mark_3 = int (input ('Enter third result:  '))
mark_4 = int (input ('Enter fourth result: '))
mark_5 = int (input ('Enter fifth result:  '))

total_marks = mark_1 + mark_2 + mark_3 + mark_4 + mark_5

average_mark = total_marks / 5

print ()
print ('Final Mark for ' + name + ' is ' + str (average_mark))
```

PyCharm Demo and Question Time

Jobs

From now on, the pace is going to quicken!

Before next week:

- Read Units 0, 1 and 2 from the book.
  - (We are currently somewhere in Unit 2.)
- Implement the "Capstone Project" from the end of Unit 1.
- Practice, practice, practice!