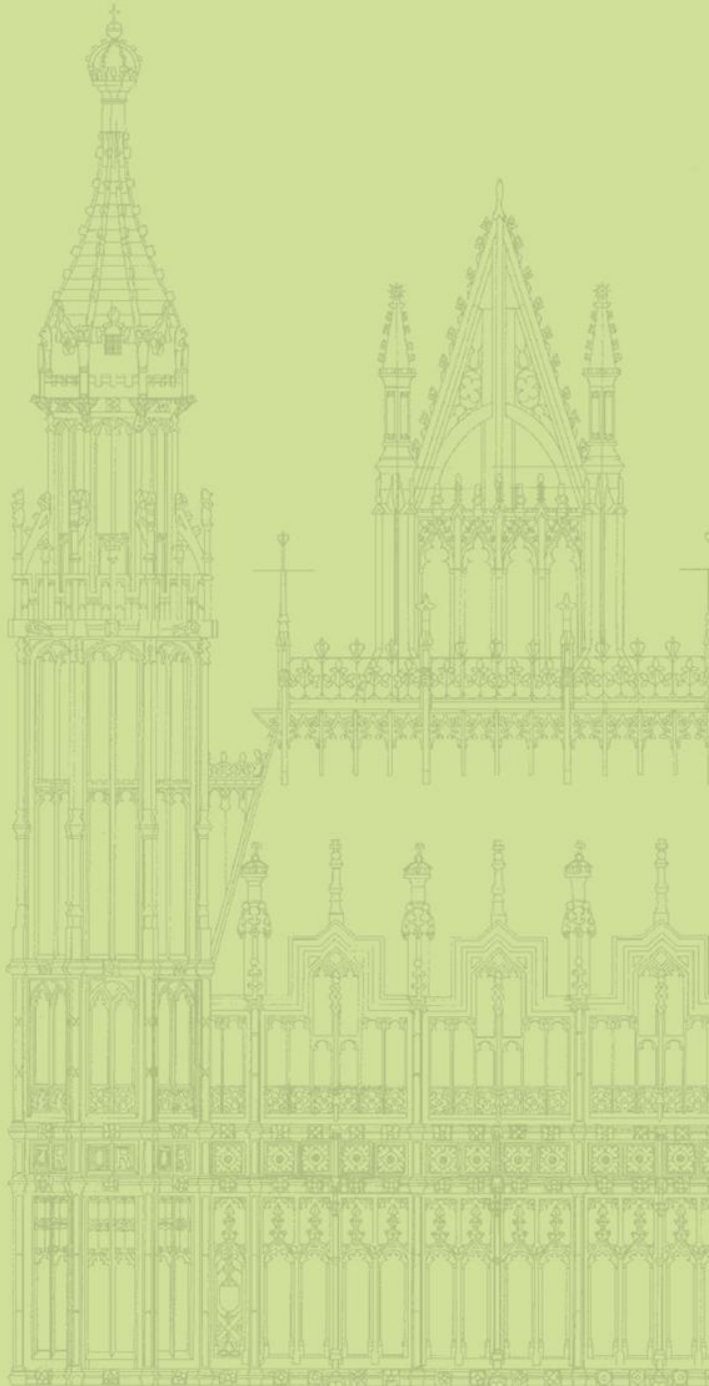
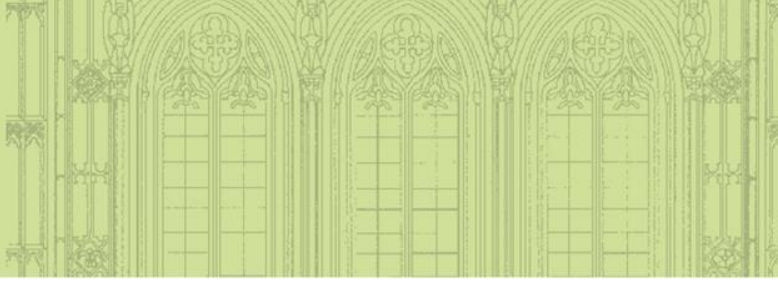
HOUSE OF COMMONS
LIBRARY

Introduction to Programming

Oliver Hawkins



HOUSE OF COMMONS
LIBRARY



Introduction to Programming

BACKGROUND TO PROGRAMMING LANGUAGES

Some languages used for data analysis

- Python
- R
- Julia
- JavaScript
- Stata

Why Python?

• Data analysis	pandas
• Statistics	numpy, scipy
• Machine learning	scikit-learn
• Databases	sqlite3 (and many others)
• Downloading	requests
• CSVs	csv
• JSON	json
• Visualisation	matplotlib, seaborn, ggplot, bokeh

Python 2 and 3

- Python 2 is older and more established
- Python 3 is newer and better
- They are **not compatible**
- Learn and use Python 3
- Use Python 2 only if necessary

Python distributions

A Python distribution is a collection of tools for working with Python …

- Python interpreter
- Python packages (libraries)
- Python shell

Some distributions: **Anaconda**, Canopy, Winpython

Before we begin ...

- This tutorial is aimed at getting you started.
- Its purpose it to introduce you to the core concepts of programming with Python.
- It is **not comprehensive**.
- It is **simplified**.
- Think of it as a map that will introduce you to the territory and help you to explore it further.

Programming is experimental

Model
How you think the program works

Theory
What effect changing the code will have

Experiment
Try changing the code

Result
Success → New theory
Failure → Revise model → New theory

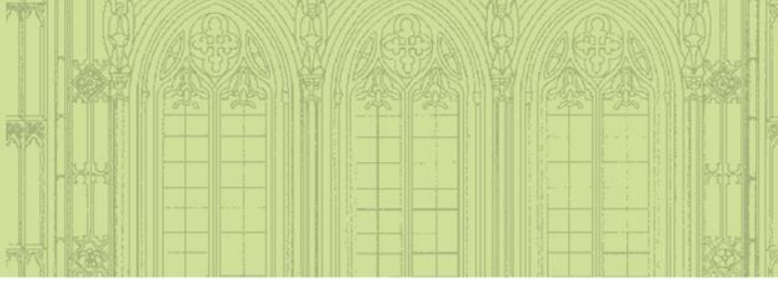
Online Python interpreter

https://repl.it

Learn by experiment

- Load the Python 3 interpreter at repl.it in a browser.
- As we go through the slides try typing the lines of code into the interpreter.
- Hit enter after each line.
- Look at the output and inspect the variables to check what is happening.

HOUSE OF COMMONS
LIBRARY



Introduction to Programming

VARIABLES, TYPES, AND OPERATORS

Variables and assignment

- A variable is a container that stores a value …

```
# Assigning a value to a variable
a = 1
b = "hello"

# Print the contents of a variable with print()
print(a)
print(b)
```

Data types

- Python is a dynamically-typed language …

```
# Basic data types
a = 1          # int          (integer)
b = 1.0        # float        (floating point number)
c = "1"        # str          (string)
d = True       # bool         (boolean)

# Use type() to find out a variable's type
type(a)
```

Operators

- You can combine variables with operators …

```
# Addition operator
x = 1
y = 2
z = x + y      # z = 3
```

Operators

- The arithmetic operators are …

+	addition	2 + 2
-	subtraction	2 - 2
*	multiplication	2 * 2
/	division	2 / 2
**	exponent	2**2
%	modulus	2 % 2

Operators

- Operators do different things depending on the variable type …

```
# Concatenation operator
x = "1"
y = "2"
z = x + y      # z = "12"
```

Operators and types

- If you combine variables of different types sometimes Python will know what to do …

```
3 / 2.0      # This returns 1.5
3 / 2        # This also returns 1.5
```

- And sometimes it won't …

```
1 + "2"      # This returns an error
```

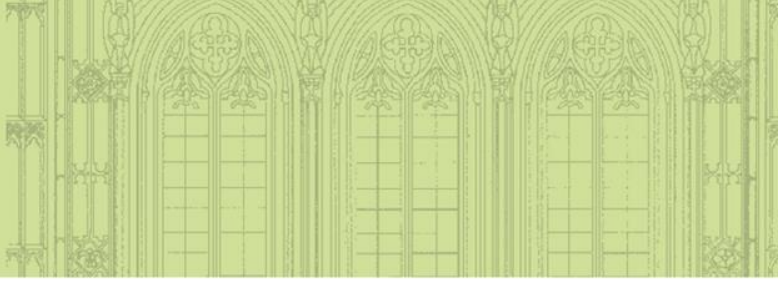
Casting variables

- You can convert a variable between types by **casting** it to a different type …

```
# Cast a string to an integer
1 + int("2")        # Returns 3

# Cast an integer to a string
str(1) + "2"        # Returns “12”
```

HOUSE OF COMMONS
LIBRARY



Introduction to Programming

COLLECTIONS

Lists

- A list is an ordered collection of data items.

```
# Declaring a list
a = [1, 2, 3, 4, 5]

# Use len() to get the number of items
len(a)
```

Lists

- You can get a particular item from a list by using its index. Lists are indexed from **zero** …

```
# Indexing a list
a[0]   # Returns 1
a[1]   # Returns 2
a[5]   # Returns an error: out of range
```

Lists

- You can add items to a list with append …

```
# Create an empty list
b = []

# Append an item to it
b.append(1)
```

Lists

- You can remove items from lists using the del statement …

```
# Remove the item at the given index
del a[0]
```

Tuples

- Tuples are another type of sequence. They are similar to lists in that they can be indexed by position, but they **cannot be changed**.

- Declare a tuple with a comma separated list …

```
t = 1,2,3,4          # Declare a tuple
print(t[0])          # Print the first item
```

Unpacking

- Lists and tuples can be unpacked into individual variables …

```
myList = [1,2]
myTuple = 3,4
a, b = myList       # a = 1, b = 2
c, d = myTuple      # c = 3, d = 4
```

Dictionaries

- Dictionaries are a collection of data items stored as key:value pairs.

```
# Declaring a dictionary
d = {"name": "Alice", "age": 5}

# Use len() to get the number of k:v pairs
len(d)
```

Dictionaries

- You can get a particular item from a dictionary by indexing with the relevant key …

```
# Indexing a dictionary
d["name"]        # Returns "Alice"
d["age"]         # Returns 5
d["hobby"]       # Returns an error:
                 # key does not exist
```

Dictionaries

- You can add items to a dictionary by assigning a new key and value …

```
# Create an empty dictionary
e = {}

# Add an item to it
e["name"] = "Ben"
e["age"] = 2
e["hobby"] = "Lego"
```

Dictionaries

- You can remove items from dictionaries using the del statement …

```
# Remove the item with the given key
del e["hobby"]
```

Values and references

- Look at this difference between how numbers and lists work …

```
# Numbers
a = 2
b = a
a = a + 1

# a = 3
# b = 2
```

```
# Lists
a = [1, 2, 3]
b = a
a.append(4)

# a = [1, 2, 3, 4]
# b = [1, 2, 3, 4]
```

Values and references

- Number variables appear to receive a **value**
  - a number is copied from one variable to another
- List variables appear to receive a **reference**
  - a reference to the list is copied from one variable to another, **but both references point to the same list**

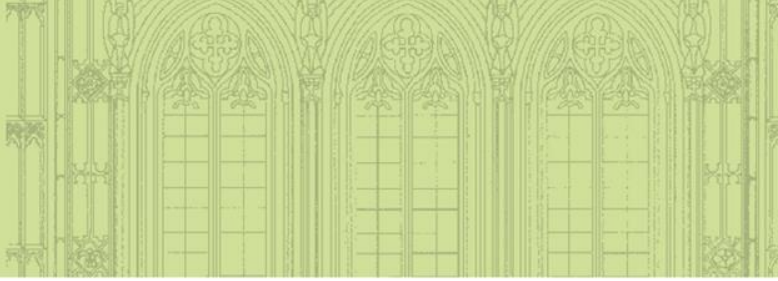
Values and references

- Some types behave as **values** and some behave as **references**.

- Generally speaking …
  - More basic types behave as values*
  - More complicated types behave as references

* In practice basic types also use references. But as numbers, strings, and tuples cannot be changed they behave like values, because when you modify them you create new instances.

HOUSE OF COMMONS
LIBRARY



Introduction to Programming

CONTROL FLOW

Conditional statements

- Use an **if statement** to execute code only in certain circumstances …

```
# If x is True print "hello"
if x:
        print("hello")
```

- Note that the print call is indented, using spaces or tabs. Whitespace is used to structure the code.

Conditional statements

- If statements execute code if the condition evaluates to True. You can use these operators to compare values in an if statement …

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

Conditional statements

- Some example comparisons …

```
x < 10
y > 0
z < len(myList)
s == "a particular string"
s != "a particular string"
```

- But the condition in an if statement doesn't have to be a comparison, **it can be any expression that evaluates to True or False**.

Conditional statements

- You can test multiple conditions in an if statement using the **and** operator and the **or** operator …

```
if x > 0 and x < 10:
if s == "this string" or s == "that string":
```

Conditional statements

- Use an **if-else statement** to execute different code in mutually exclusive circumstances …

```
if x < 0:
        # Do this if x is a negative number
else:
        # Do this if x is zero or positive
```

Conditional statements

- You can chain conditional tests using **elif** …

```
if x == 0:
        # Do this if x is zero
elif x == 1:
        # Do this if x is one
else:
        # Do this if x is any other value
```

Loops

- Loop statements let you execute the same block of code over and over, until a condition is met.
- Loops are typically used to process data one value (or one group of values) at a time.
- There are different types of loops: **while** loops and **for** loops.

While loops

- This is a **while** loop …

```
x = 0                   # Set x to zero
while x < 10:           # Is x less than ten? If so …
    print(x)            # Print x
    x = x + 1           # Add one to x
```

- The code inside the loop is run until x is greater than nine. So the code prints integers from zero to nine.

While loops

- You can use a **while** loop to loop through a list …

```
myList = ["a", "b", "c"]   # Declare a list
i = 0                      # Set i to zero
n = len(myList)            # Set n to list length
while i < n:               # Is i less than n?
        print(myList[i])   # Print item at index i
        i = i + 1          # Add one to i
```

For loops

- But it's much easier to use a **for** loop in Python …

```
myList = ["a", "b", "c"]   # Declare a list
for x in myList:           # For each item in list
    print(x)               # Print item
```

- **for** loops give you each item in a sequence. You can use them to loop through dictionaries too by getting the dictionary data as a sequence.

For loops

- You can get data from **dictionaries** as sequences using **keys(), values()** and **items()** ...

```
d = {"name": "Alice", "age": 5}
d.keys()      # ["age", "name"]
d.values()    # [5, "Alice"]
d.items()     # [('age', 5), ('name', 'Alice')]
```

- The **items()** method gives you the key:value pairs as a list of **tuples**.

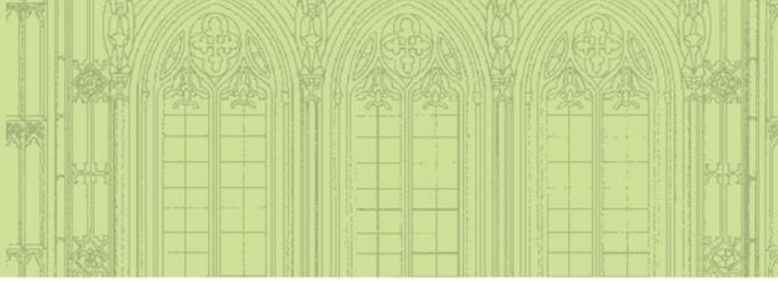
For loops

- This code loops over all the keys in a dictionary and prints the key and value for each key …

```
for key in d.keys():                # For each key
    print(key, d[key])              # Print k, v pair
```

- Dictionaries are not intrinsically ordered. If you need to loop over a dictionary's items in a particular order, sort the keys accordingly.

HOUSE OF COMMONS
LIBRARY



Introduction to Programming

FUNCTIONS

Functions

- Functions are discrete units of code that perform a particular task. We have already seen some built-in functions.
  - The **print()** function prints the value of a variable to the output: print("a string")
  - The **len()** function returns the length of a list, tuple, or dictionary: len(myList)

Functions

- You can define your own functions with the **def** keyword …

```
def myFunction():
    print("this is my function")
```

- Now when you call myFunction() in the interpreter it will print: "this is my function"

Functions

- Functions can take **arguments**. These are values you can pass into the function when you call it.

```
def hello(name):
    print("hello " + name)
```

- Now call: hello("world")

Functions

- You can **return** values from a function …

```
def double(x):
    y = x * 2      # Multiply x by two
    return y       # Return the new value
```

- Now call: double(4)

Putting it all together

- This function takes a list of numbers and returns a list containing only the even numbers …

```
def getEvenNumbers(numbers):
        evens = []
        for x in numbers:
                if x % 2 == 0:
                        evens.append(x)
        return evens
```

Now call: `getEvenNumbers([1,2,3,4,5])`

Congratulations

You just did some programming!

^_^

Where to learn more

- The Python Tutorial (official introduction)

https://docs.python.org/3.4/tutorial/index.html

- Code Club

https://www.codeclubprojects.org/en-GB/python/

- Code Academy

https://www.codecademy.com/

- Future Learn

https://www.futurelearn.com/courses/learn-to-code