Week 3
Modules & objects

Importing

In Python, it is common to import functionality from other files. There are two main ways to import stuff.

First, you can import an entire module, then use functionality from within that module by writing `module_name.function_name`.

```
import math
print(math.log(2))
```

```
0.6931471805599453
```

Alternatively, you can import a single function from a module.

```
from math import log
print(log(2))
```

```
0.6931471805599453
```

The `as` command allows you to rename things you are importing:

```
import math as maths
print(maths.log(2))
```

```
0.6931471805599453
```

```
from math import log as ln
print(ln(2))
```

```
0.6931471805599453
```

We now look at some of the most commonly imported modules.

`math`

The `math` module includes many mathematical functions, including `math.cos`, `math.sin`, `math.tan`, `math.log`, `math.floor`, `math.factorial`, `math.gcd`, `math.sinh` and `math.cosh`.

`random`

The `random` module includes functions to create random numbers.

`random.random` generates a random number between 0 and 1.

```
from random import random
print(random())
```

```
0.6931471805599453
```

`random.choice` chooses a random item from a list, dictionary, or tuple.

```
import random
a = [2,3,10,8]
print(random.choice(a))
```

```
3
```

`random.shuffle` reorders a list randomly.

```
import random
a = [2,3,10,8]
random.shuffle(a)
print(a)
```

```
[8, 10, 2, 3]
```

`random.randrange` picks a random number from a range. The inputs of this function are interpreted in the same way as `range`.

```
from random import randrange
print(randrange(2,10))
```

```
4
```

`random.gauss` samples a normal distribution. The first input of this function is the mean, the second is the standard deviation.

```
import random
print(random.gauss(0,1))
print(random.gauss(2,0.1))
```

```
1.1453651582510647
1.8077073476534284
```

1. Generate a random real number in $[0, 10)$.

2. Write a Python script that simulates the rolling of two (fair) dice.

itertools

`itertools` contains functions that are useful when iterating through things.

`itertools.combinations` will iterate through all ordered subsets of a given length.

```
import itertools
for i in itertools.combinations(range(5), 3):
    print(i)
```

```
(0, 1, 2)
(0, 1, 3)
(0, 1, 4)
(0, 2, 3)
(0, 2, 4)
(0, 3, 4)
(1, 2, 3)
(1, 2, 4)
(1, 3, 4)
(2, 3, 4)
```

`itertools.combinations_with_replacement` will iterate through all subsets of a given length with repeats allowed.

```
import itertools
for i in itertools.combinations_with_replacement(range(5), 3):
    print(i)
```

```
(0, 0, 0)
(0, 0, 1)
(0, 0, 2)
(0, 0, 3)
(0, 0, 4)
(0, 1, 1)
(0, 1, 2)
(0, 1, 3)
(0, 1, 4)
(0, 2, 2)
(0, 2, 3)
(0, 2, 4)
(0, 3, 3)
(0, 3, 4)
(0, 4, 4)
(1, 1, 1)
(1, 1, 2)
(1, 1, 3)
...
```

`itertools.permutations` will iterate through all ordered subsets of a given length.

```
import itertools
for i in itertools.permutations(range(5), 3):
    print(i)
```

```
(0, 1, 2)
(0, 1, 3)
(0, 1, 4)
```

```
(0, 2, 1)
(0, 2, 3)
(0, 2, 4)
(0, 3, 1)
(0, 3, 2)
(0, 3, 4)
(0, 4, 1)
(0, 4, 2)
(0, 4, 3)
(1, 0, 2)
(1, 0, 3)
(1, 0, 4)
...
```

`itertools.product` will iterate through the product of two or more iterables.

```
import itertools
for i in itertools.product(range(5), "abc"):
    print(i)
for i in itertools.product(range(5), "abc", "def"):
    print(i)
```

```
(0, 'a')
(0, 'b')
(0, 'c')
(1, 'a')
(1, 'b')
(1, 'c')
(2, 'a')
(2, 'b')
(2, 'c')
(3, 'a')
(3, 'b')
(3, 'c')
(4, 'a')
(4, 'b')
(4, 'c')
(0, 'a', 'd')
(0, 'a', 'e')
(0, 'a', 'f')
(0, 'b', 'd')
(0, 'b', 'e')
(0, 'b', 'f')
(0, 'c', 'd')
(0, 'c', 'e')
(0, 'c', 'f')
(1, 'a', 'd')
(1, 'a', 'e')
(1, 'a', 'f')
...
```

`itertools.repeat` will repeat the object a given number of times (or forever if no number is given).

```
import itertools
for i in itertools.repeat(5, 3):
    print(i)
```

```
5
5
```

```
5
```

WARNING

The next code example leads to an infinite loop.

```python
import itertools
for i in itertools.repeat("Hello There"):
    print(i)
```

```
Hello There
Hello There
Hello There
Hello There
Hello There
Hello There
Hello There
...
```

3. A bag contains 3 red balls, 2 yellow balls, and 2 blue balls. I pick a ball then pick a second ball (without replacing the first). Write a Python script that prints out all the possible combinations that I could pick. What is the probability that I pick two different colours?

4. A bag contains 3 red balls, 2 yellow balls, and 2 blue balls. I pick a ball, replace it, then pick a second ball. Write a Python script that prints out all the possible combinations that I could pick. What is the probability that I pick two different colours?

Other common imports

1. `datetime` - This module can be used to deal with dates and times.
2. `json` - This module can be used to import and export objects using json notation.
3. `urllib` - This module can be used to work with URLs.
4. `numpy`* - This module allows you to create vectors and contains many functions to work with these vectors.
5. `scipy`* - This module contains many functions from linear algebra, optimisation, etc.
6. `matplotlib`* - This module allows you to plot graphs.
7. `pandas`* - This module provides data analysis tools.
8. `sympy`* - This module allows you to do symbolic albgebra, such as expanding brackets, differentiation, and evaluation.

The libraries marked with an asterisk are not included with Python by default so must be installed before use. Next week, we will be using `numpy` and `matplotlib`; instructions for installing these will be distributed shortly after this week's lecture.

Using Google/Bing/DuckDuckGo

If you want to use one of the modules above, or find a module that can do something else, the best thing to do it to Google it. When Googling for Python help, you should bear a few things in mind:

- We are using Python 3, not Python 2. The way some modules/libraries work has changed since Python 2 so it is often worth including "Python 3" in your search terms.
- Some websites are not very helpful. If you don't understand what the website says, give up and try a different one.
- Most Python help searches will lead you to StackOverflow. On StackOverflow, a user asks a question then other users reply.
  - Scroll down past the question and the comments on it to the answers.
  - The top answers are usually best, but if the first one doesn't make sense, move on to the next.

5. The URL `https://api.tfl.gov.uk/stopPoint/940GZZLUEUS/arrivals?mode=tube` contains live underground data for Euston station. Using Google, Bing, or DuckDuckGo, work out how to load the json from the URL into Python then print out the next trains to arrive.

Importing your own code

As well as loading modules that you have installed (or that are included with Python by default), you can also import your own code.

If you tell Python to `import apple`, it will:

1. Try to import a file or folder called `apple` in the directory you are currently in.
2. If there is no file/folder called `apple`, it will load an installed module called `apple`.
3. If there is no file/folder or module called `apple`, it will throw an `ModuleNotFoundError` .

Write the following code in `apple.py`.

```python
def f():
    print("it works")
```

Then make a new file (in the same directory), and write the following.

```
from apple import f
f()
```

```
it works
```

We are going to write a module containing functions that create lists of certain types of number.

Write the following and save it as `listmaker.py`.

```
def squares(n):
    "Returns a list of square numbers between 1 and n."
    squares = []
    i = 1
    while i**2 < n:
        squares.append(i**2)
        i += 1
    return squares

def primes(n):
    "Returns a list of prime numbers between 1 and n."
    primes = []
    for i in range(2,n):
        a = True
        for p in primes:
            if i%p == 0:
                a = False
                break
            if p**2 > i:
                break
        if a:
            primes.append(i)
    return primes

def cubes(n):
    "Returns a list of cube numbers between 1 and n."
    pass

def even(n):
    "Returns a list of even numbers between 1 and n."
    pass

def fibonacci(n):
    "Returns a list of Fibonacci numbers between 1 and n."
    pass
```

6. Write the last three functions.

We are now going to use our module to see which numbers below 100 cannot be written as the sum of a prime number and a square number. Write the following in a new file in the same directory as `listmaker.py`.

```
import listmaker
from itertools import product

squares = listmaker.squares(100)
```

```
primes = listmaker.primes(100)

possible = {}
for i in range(1,100):
    possible[i] = False

for i,j in product(squares, primes):
    if i+j < 100:
        possible[i+j] = True

for i,j in possible.items():
    if not j:
        print(i)
```

```
1
2
5
10
13
25
31
34
37
58
61
64
85
91
```

7. Write a Python script that prints the numbers below 100 that are not equal to a Fibonacci number plus a prime number. Compare your results to `http://oeis.org/A132146` to check them.

Objects

As well as importing functions, you can also import object types. To write an object type, you can use the keyword `class`.

The following code is the start of a fraction object type. Save it as `frac.py`.

```
class Fraction:
    def __init__(self, num, denom):
        self.num = num
        self.denom = denom

    def as_decimal(self):
        return self.num / self.denom

    def __add__(self, other):
        denom = self.denom * other.denom
        num = self.num*other.denom + other.num*self.denom
        return Fraction(num, denom)

    def __str__(self):
        return str(self.num) + "/" + str(self.denom)
```

It can then be used by writing the following in a file in the same directory.

```
from frac import Fraction
half = Fraction(1, 2)
third = Fraction(1, 3)
print(half+third)
print(half.as_decimal())
```

```
5/6
0.5
```

Let's look at this line by line.

The first line imports the `Fraction` class from `frac.py`.

```
from frac import Fraction
```

The second line creates a `Fraction` object equal to 1/2. When the object is created, the function `__init__` is run with inputs `half`, `1`, and `2`. The first input of a function in a class is always the object itself.

```
half = Fraction(1, 2)
```

The third line creates another `Fraction` object equal to 1/3.

```
third = Fraction(1, 3)
```

The fourth line print the sum of the two fractions. To add the two fractions, the `__add__` function is called. To print the result, the `__str__` function is called.

```
print(half+third)
```

the fifth line prints the result of the `as_decimal` function.

```
print(half.as_decimal())
```

A fully functional `Fraction` class can be imported from the Python module `fractions` (included with Python by default):

```
from fractions import Fraction
half = Fraction(1,2)
third = Fraction(1,3)
print(half+third)
print(float(half))
```

```
5/6
0.5
```

Debugging and error handling

When there is an error in your Python code, Python will tell you what type of error/exception is happening and which line is causing it. The following code is written in `err.py`.

```
a = 2
for b in range(-5,6):
    print(a/b)
```

```
-0.4
-0.5
-0.6666666666666666
-1.0
-2.0
Traceback (most recent call last):
  File "err.py", line 3, in <module>
    print(a/b)
ZeroDivisionError: division by zero
```

Running this file leads to the above error. This tells us that a `ZeroDivisionError` has occured on line 3. When you get an error, you can use this information to correct the error.

Sometimes it is useful to get Python to ignore an exception and carry on running. You can do this using `try` and `except`. The following code will print `a / b`, but if computing this leads to a `ZeroDivisionError`, it will instead print `"Ignoring division by zero"`.

```
a = 2
for b in range(-5,6):
    try:
        print(a/b)
    except ZeroDivisionError:
        print("Ignoring division by zero")
```

```
-0.4
-0.5
-0.6666666666666666
-1.0
-2.0
Ignoring division by zero
2.0
1.0
0.6666666666666666
0.5
0.4
```

Writing `except` without an exception type will catch all possible exceptions.

```
a = 2
data = []
for b in range(-5,6):
    try:
        print(a/b)
        print(data[a])
    except:
        print("Ignoring error")
```

```
-0.4
Ignoring error
-0.5
Ignoring error
-0.6666666666666666
Ignoring error
```

```
-1.0
Ignoring error
-2.0
Ignoring error
Ignoring error
2.0
Ignoring error
1.0
Ignoring error
0.6666666666666666
Ignoring error
0.5
Ignoring error
0.4
Ignoring error
```

The most common exception types in Python are:

- `ZeroDivisionError`
- `AttributeError`
- `ImportError`
- `ModuleNotFoundError`
- `IndexError`
- `KeyError`
- `MemoryError`
- `NameError`
- `SyntaxError`
- `IndentationError`
- `ValueError`
- `KeyboardInterrupt`

8. For each of the exception types above, write some Python code that causes them.

Putting it all together

Using all the Python commands you have learned in the three weeks so far, do the following questions.

9. Write a Python script using the `random` module that simulates the rolling of three (fair) dice. Use a for loop to simulate rolling three dice 1000 times. What is the probability that the total of the three dice is greater than 8?

10. Write a Python script using the `itertools` module that iterates through all possible outcomes of rolling three (fair) dice. What is the probability that the total of the three dice is greater than 8?

11. Using the `listmaker` module that we wrote above, print all the numbers below 100 that cannot be written as the sum of a square number and a cube number.

12. What is the largest number that cannot be written as $13a + 119b$, where a and b are positive integers or 0?

13. In January this year, we had a "thirdsday": the third day of the month was on a Thursday. Using the `datetime` module (and Googling how to use it) write a Python script that prints out the months in which there will be a thirdsday in 2019, 2020, 2021, and 2022.

14. In the UK, we have 1p, 2p, 5p, 10p, 20p, 50p, £1 and £2 coins. What is the lowest amount of money that cannot be made using fewer than 5 coins?

15. Alice flips 6 (fair) coins and Bob flips 5 (fair) coins. Alice wins if she gets more heads than Bob. Write a Python script the simulates this game. Simulate 1000 games. What is the probability that Alice wins?

Assessed question

When you have completed this question, you should show your code to one of the helpers in your class.

16. It was proposed by Christian Goldbach that every odd composite number can be written as the sum of a prime and twice a square:

$$\begin{aligned} 9 &= 7 + 2 \times 1^2 \\ 15 &= 7 + 2 \times 2^2 \\ 21 &= 3 + 2 \times 3^2 \\ 25 &= 7 + 2 \times 3^2 \\ 27 &= 19 + 2 \times 2^2 \\ 33 &= 31 + 2 \times 1^2 \end{aligned}$$

It turns out that the conjecture was false. What is the smallest odd composite that cannot be writen as the sum of a prime and twice a square?