Lab 6.2 (Deadline Monday 28 February 23:59)

Function arguments and parameters

- Without running any code, predict the output of each of the following invocations of the `arithmetic()` function.
- If any invocation raises an error, explain the error.

```
#!/usr/bin/env python3

def arithmetic(p, q, r=5, s=2):
    return r - p + q + s

def main():

    print(arithmetic(1, 2, 5, 6))

    print(arithmetic(3, 4, 5))

    print(arithmetic(3, 4))

    print(arithmetic(3, 4, s=3))

    print(arithmetic(s=5, q=4, p=2, r=1))

    print(arithmetic(q=2, p=4, 6))

    print(arithmetic(6, r=2, p=4))

    print(arithmetic(p=2, q=4, s=6))

    print(arithmetic(p=5, 2, 5))


if __name__ == '__main__':
    main()
```

- Once you have a set of predictions, run the code one call at a time to verify your answers are correct.
- There is no Einstein marker for this exercise.

Perfect numbers

- Write a *function* `sum_factors()` which specifies an integer parameter `n` (assumed positive) and returns the sum of the factors of `n`.
- The factors of an integer `n` are the positive integers that exactly divide `n`, not including `n` itself.
- For example, the factors of 12 are 1, 2, 3, 4, and 6, and so `sum_factors(12)` should return 16.
- Write a boolean-valued *function* `is_perfect()` which specifies an integer parameter `n` (assumed positive) and returns whether `n` is perfect.
- A positive number is perfect if it is equal to the sum of its factors.
- For example 28 is perfect because its factors are 1, 2, 4, 7, 14, and, 28 = 1 + 2 + 4 + 7 + 14.

- Write a program called *perfect_062.py* which reads a list of positive integers from `stdin` (one per line) and for each one prints `True` if it is a perfect number and `False` otherwise.
- Obviously, you want to make good use of the two functions you developed above.

```
$ cat perfect_stdin_00_062.txt
1
12
33550336
10
28
```

```
$ python3 perfect_062.py < perfect_stdin_00_062.txt
False
False
True
False
True
```

The mutable default parameter value trap

- The behaviour of the code below is not as the programmer intended.
- Identify and make sure you understand the problem.
- Fix the code and call it *mutable_062.py*.

```
#!/usr/bin/env python3

# Append l1 to l2. If l2 not supplied default to empty list.
def append2list(l1, l2=[]):
    for i in l1:
        l2.append(i)
    return l2

def main():
    list1 = ['fly', 'spider']
    nlist = append2list(list1)
    # nlist should be ['fly', 'spider']
    print(nlist)

    list2 = ['lion']
    nlist = append2list(list2, ['antelope'])
    # nlist should be ['antelope', 'lion']
    print(nlist)

    list3 = ['fox', 'chicken']
    nlist = append2list(list3)
    # nlist should be ['fox', 'chicken']
    print(nlist)


if __name__ == '__main__':
    main()
```

```
['fly', 'spider']
['antelope', 'lion']
['fly', 'spider', 'fox', 'chicken']
```

Overlapping circles

- Write a function called `overlap()` that returns `True` if two circles overlap and `False` otherwise.
- A circle is defined by its centre's x and y coordinates and its radius.
- Thus our function specifies six parameters in the following order: x1, y1, r1, x2, y2, r2 (you must use these parameter names and in the specified order when defining your function).
- By default, circles are centred at (0, 0) and radii are 1.
- Put your function in a module called *`circle_062.py`*.

```
#!/usr/bin/env python3

from circle_062 import overlap

def main():
    print(overlap())
    print(overlap(10))
    print(overlap(10,10))
    print(overlap(10,10,10))
    print(overlap(10,0,10))
    print(overlap(10,0,1,8,0,1))
    print(overlap(10,0,1,8,0,2))


if __name__ == '__main__':
    main()
```

```
True
False
False
False
True
False
True
```

- Hint: A formula that works out the distance between two points might be handy.

Quadratic roots

- The roots of the quadratic:

$$f(x) = ax^2 + bx + c$$

  are given by:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Write a program called *`roots_062.py`* that reads a, b, and c from `stdin` (one set per line) and passes them to a function that computes and returns the corresponding roots.
- Should no real roots exist then the program should print 'None'.
- Otherwise both roots should be printed.

```
$ cat roots_stdin_00_062.txt
1 0 -1
```

```
1 1 -2
1 1 2
1 2 1
1 6 5
1 5 6
2 -2 -12
```

```
$ python3 roots_062.py < roots_stdin_00_062.txt
r1 = 1.0, r2 = -1.0
r1 = 1.0, r2 = -2.0
None
r1 = -1.0, r2 = -1.0
r1 = -1.0, r2 = -5.0
r1 = -2.0, r2 = -3.0
r1 = 3.0, r2 = -2.0
```