In [7]:

```python
from time import perf_counter


class Counter:
    def __repr__(self):
        return f"Counter(\"{self._func.__name__}\")"

    def __init__(self, func, *values):
        self._func = func
        self._values = values
        self._inners = []
        self._outers = []
        self._t_buff = perf_counter()

    def reset_buffer(self):
        self._t_buff = perf_counter()

    def log_inner(self):
        t = perf_counter()
        self._inners.append(t - self._t_buff)
        self._t_buff = t

    def log_outer(self):
        t = perf_counter()
        self._outers.append(t - self._t_buff)
        self._t_buff = t

    def time(self):
        return time(self._func, self, *self._values)


def some_complex_calculation(counter: Counter, *values):
    """
    Creates a sum using an algorithm which adds a[n] to a sum f a[n - 1] times, where n is a real number greater than 0 and a is the array of values passed to the function

    :counter: a counter which can be used to make time logs from outer an inner loops
    """
    v_sum = 0
    l_buff = 0
    counter.reset_buffer()
    for i, v in enumerate(values):
        counter.reset_buffer()
        for _ in range(l_buff):
            v_sum += v
        counter.log_inner()
        l_buff = v
    counter.log_outer()

def time(func, *args) -> float:
    """
    Times a function call with the given arguments then returns the perf_counter() difference from
    before call and after call
    """
    start = perf_counter()
    some_complex_calculation(*args)
    count = perf_counter() - start
    print(f"{func.__name__}({", ".join(map(repr, args))}) took {count})")

    return count


test_counter = Counter(some_complex_calculation, 1, 2, 3, 4, 5)
test_counter.time()


from random import randint


for _ in range(10):
    Counter(some_complex_calculation, *(randint(1, 10) for _ in range(randint(5, 10)))).time()
```

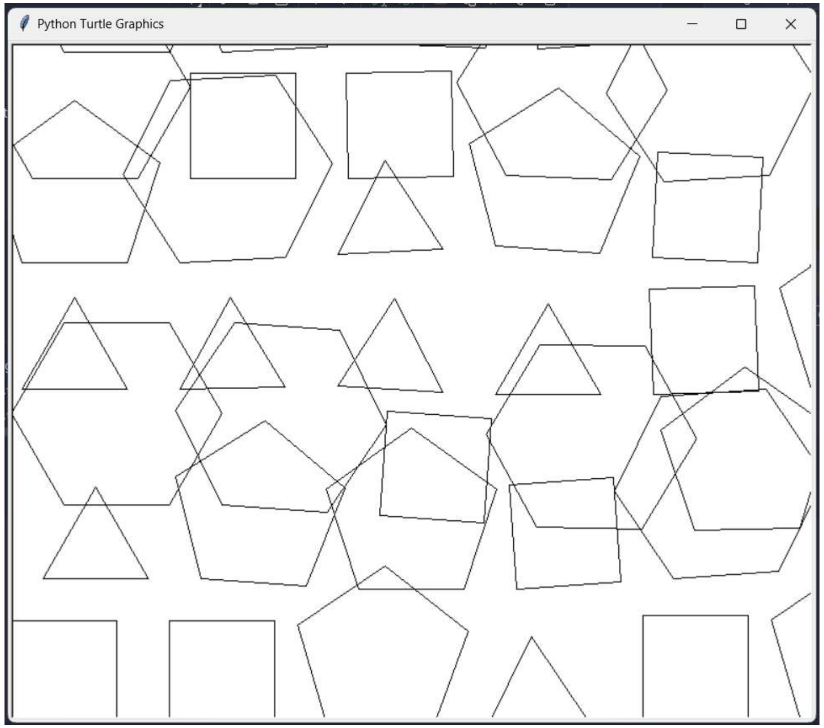
```
some_complex_calculation(Counter("some_complex_calculation"), 1, 2, 3, 4, 5) took 1.0799998563015833e-05)
some_complex_calculation(Counter("some_complex_calculation"), 10, 9, 4, 3, 4, 5, 6, 9, 10) took 8.699998943484388e-06)
some_complex_calculation(Counter("some_complex_calculation"), 10, 6, 5, 6, 7, 4, 6, 1, 5, 7) took 5.899999450775795e-06)
some_complex_calculation(Counter("some_complex_calculation"), 10, 7, 4, 2, 6, 5, 6) took 4.19999923906289e-06)
some_complex_calculation(Counter("some_complex_calculation"), 7, 2, 6, 3, 5, 7, 5, 4, 4) took 4.800000169780105e-06)
some_complex_calculation(Counter("some_complex_calculation"), 4, 3, 4, 10, 5, 5, 1, 5, 7) took 4.600000465870835e-06)
some_complex_calculation(Counter("some_complex_calculation"), 4, 2, 3, 9, 7, 4, 3, 5, 9) took 4.499999704421498e-06)
some_complex_calculation(Counter("some_complex_calculation"), 7, 10, 1, 5, 2, 2) took 3.1999989005271345e-06)
some_complex_calculation(Counter("some_complex_calculation"), 5, 7, 10, 1, 2, 7) took 3.099999958067201e-06)
some_complex_calculation(Counter("some_complex_calculation"), 7, 6, 5, 1, 10) took 2.7999994927085936e-06)
some_complex_calculation(Counter("some_complex_calculation"), 7, 4, 8, 9, 4, 6, 3) took 3.6999990697950125e-06)
```

In [9]:

```python
from random import randint
from time import perf_counter
from turtle import Screen, Turtle

screen = Screen()
turtle = Turtle()


def reset_turtle():
    turtle.penup()
    turtle.setpos((0 - screen.window_width()) // 2, (0 - screen.window_height()) // 2)
    turtle.setheading(0)


turtle.shape("turtle"), turtle.speed(7)
reset_turtle()


def shape(sides, step=100):
    def _():
        start = perf_counter()
        turtle.pendown()
        for _ in range(sides):
            turtle.forward(step)
            turtle.left(360 // sides)
        turtle.penup()
```

```python
        count = perf_counter() - start
        print("%s sided shape took %s" % (sides, count))
        return step

    return _


heading = 0


for y in range(1, 11):
    last_step = 0
    for _ in range(6):
        last_step = shape(randint(3, 6))()
        turtle.forward(last_step * 1.5)
        turtle.setheading(heading + randint(0, 8) - 4)

    reset_turtle()

    turtle.forward(randint(1, 5) * 10)
    turtle.left(90)
    turtle.forward(last_step * y + randint(1, 5) * 10)
    turtle.right(90)

    turtle.setheading(0)

screen.exitonclick()
```

```
4 sided shape took 0.5933393999985128
4 sided shape took 0.595412499998929
6 sided shape took 0.6877696999999898
4 sided shape took 0.6102804000001925
4 sided shape took 0.5944256000002497
4 sided shape took 0.5936985000007553
4 sided shape took 0.5952634000004764
4 sided shape took 0.5916042000008019
5 sided shape took 0.6567388999992545
5 sided shape took 0.6585513999998511
5 sided shape took 0.6603314999993017
6 sided shape took 0.699409700000615
5 sided shape took 0.6537995999988198
4 sided shape took 0.5946595000004891
3 sided shape took 0.5165381999995589
6 sided shape took 0.7709515000005922
4 sided shape took 0.6106772999992245
4 sided shape took 0.5885065999991639
6 sided shape took 0.6906288000009226
3 sided shape took 0.5036380999990797
3 sided shape took 0.5006379999995261
3 sided shape took 0.50228000000061
5 sided shape took 0.6584853999993356
3 sided shape took 0.49851660000058473
3 sided shape took 0.5003582999997889
6 sided shape took 0.6872835000012856
5 sided shape took 0.656813499999771
6 sided shape took 0.6859237000007852
5 sided shape took 0.6597927000002528
4 sided shape took 0.5955731999983982
5 sided shape took 0.6571966999999859
6 sided shape took 0.6931939999994938
4 sided shape took 0.5939628999985871
4 sided shape took 0.5914489999995567
4 sided shape took 0.5943976999988081
4 sided shape took 0.5948184999997466
6 sided shape took 0.6859370000001945
6 sided shape took 0.6875827000003483
4 sided shape took 0.5905492999991111
4 sided shape took 0.5947145999998611
3 sided shape took 0.49893540000084613
3 sided shape took 0.4959197000007407
5 sided shape took 0.6504512999999861
3 sided shape took 0.4975524999990739
5 sided shape took 0.6574381000009453
6 sided shape took 0.6862667999994301
6 sided shape took 0.6848602999998548
4 sided shape took 0.5913661000013235
4 sided shape took 0.596433499998966
6 sided shape took 0.6876730999993015
3 sided shape took 0.497919000001275
3 sided shape took 0.49957700000049954
3 sided shape took 0.5012299999998504
4 sided shape took 0.5938977999994677
3 sided shape took 0.5127811999991536
6 sided shape took 0.6879152999990765
6 sided shape took 0.6856157000001986
5 sided shape took 0.654628700000103
6 sided shape took 0.685850500000015
5 sided shape took 0.6586675999988074
```

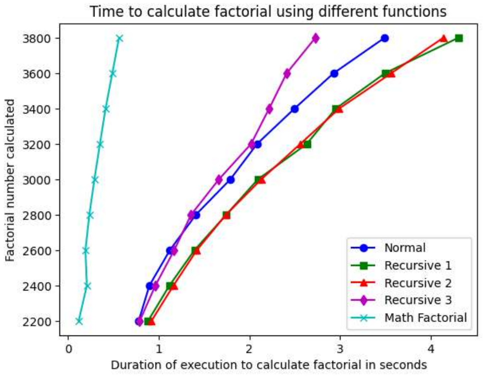


In [12]:

```
# Python code to see how fast different solutions for calculating factorial are.

import math
import sys
import time

import matplotlib.pyplot as pyplot


def fact1(n):
    # Naive method for calculating factorial of n
    # Creates a product for numbers between 2 and n with n being the number to get the factorial of
    # unless n is 1 in which case returns 1
    fact = 1
    if n > 1:
        for i in range(2, n + 1):
            fact *= i
    return fact


def fact2(n):
    # Recursive method for calculating factorial of n
    # The same as the next function except unfolded into a normal function
    if n == 1:
        return n
    else:
        return n * fact2(n - 1)


def fact3(n):
    # Recursive method using lambda function
    # With the assumption that to calculate a factorial f(n) = n * !(n - 1)
    # It recursively calls itself multiplying the given argument by itself - 1 until it gets to 0
    # at which point it will return down the call stack
    f = lambda n: n * f(n - 1) if n > 1 else 1
    return f(n)


def multiply_range(n, m):
    if n == m:
        return n
    if m < n:
        return 1
    else:
        return multiply_range(n, int((n + m) / 2)) * multiply_range(int((n + m) / 2) + 1, m)


def fact4(n):
    # Recursive method using dive and conquer method.
    return multiply_range(1, n)


sys.setrecursionlimit(10 ** 6)
# when this is commented out a recursion error may occur when the recursive function is called
# because it calls itself more than the base recursion limit (999) times


def time_func(name, limit, func):
```

```python
    start_time = time.time()
    for n in range(1, limit):
        func(n)
    duration = (time.time() - start_time)
    print(f"--- time for {name} solution: %8.5f seconds ---" % duration)
    return duration


normal = []
recursive1 = []
recursive2 = []
recursive3 = []
math_factorial = []

for limit in range(2200, 4000, 200):
    normal.append(time_func("normal", limit, fact1))
    recursive1.append(time_func("recursive 1", limit, fact2))
    recursive2.append(time_func("recursive 2", limit, fact3))
    recursive3.append(time_func("recursive 3", limit, fact4))
    math_factorial.append(time_func("better", limit, math.factorial))

Y = list(range(2200, 4000, 200))

pyplot.plot(normal, Y, marker='o', label="Normal", color='b')
pyplot.plot(recursive1, Y, marker='s', label="Recursive 1", color='g')
pyplot.plot(recursive2, Y, marker='^', label="Recursive 2", color='r')
pyplot.plot(recursive3, Y, marker='d', label="Recursive 3", color='m')
pyplot.plot(math_factorial, Y, marker='x', label="Math Factorial", color='c')

pyplot.xlabel("Duration of execution to calculate factorial in seconds")
pyplot.ylabel("Factorial number calculated")

pyplot.title("Time to calculate factorial using different functions")

pyplot.legend()

pyplot.show()

# The best function (other than the builtin function) is the divide and conquer function
# because it reduces the amount of multiplication and recursive calls compared to a normal
# recursive function more as n increases
```

```
--- time for normal solution:  0.77913 seconds ---
--- time for recursive 1 solution:  0.88220 seconds ---
--- time for recursive 2 solution:  0.91974 seconds ---
--- time for recursive 3 solution:  0.79046 seconds ---
--- time for better solution:  0.11652 seconds ---
--- time for normal solution:  0.89754 seconds ---
--- time for recursive 1 solution:  1.11849 seconds ---
--- time for recursive 2 solution:  1.16296 seconds ---
--- time for recursive 3 solution:  0.96333 seconds ---
--- time for better solution:  0.20716 seconds ---
--- time for normal solution:  1.12352 seconds ---
--- time for recursive 1 solution:  1.39512 seconds ---
--- time for recursive 2 solution:  1.42054 seconds ---
--- time for recursive 3 solution:  1.17169 seconds ---
--- time for better solution:  0.19148 seconds ---
--- time for normal solution:  1.40978 seconds ---
--- time for recursive 1 solution:  1.74475 seconds ---
--- time for recursive 2 solution:  1.74290 seconds ---
--- time for recursive 3 solution:  1.35750 seconds ---
--- time for better solution:  0.23479 seconds ---
--- time for normal solution:  1.79080 seconds ---
--- time for recursive 1 solution:  2.09513 seconds ---
--- time for recursive 2 solution:  2.13201 seconds ---
--- time for recursive 3 solution:  1.66455 seconds ---
--- time for better solution:  0.29048 seconds ---
--- time for normal solution:  2.08539 seconds ---
--- time for recursive 1 solution:  2.63406 seconds ---
--- time for recursive 2 solution:  2.56262 seconds ---
--- time for recursive 3 solution:  2.01931 seconds ---
--- time for better solution:  0.35112 seconds ---
--- time for normal solution:  2.49890 seconds ---
--- time for recursive 1 solution:  2.95439 seconds ---
--- time for recursive 2 solution:  2.99227 seconds ---
--- time for recursive 3 solution:  2.21925 seconds ---
--- time for better solution:  0.41364 seconds ---
--- time for normal solution:  2.93146 seconds ---
--- time for recursive 1 solution:  3.49892 seconds ---
--- time for recursive 2 solution:  3.56766 seconds ---
--- time for recursive 3 solution:  2.41312 seconds ---
--- time for better solution:  0.48814 seconds ---
--- time for normal solution:  3.49369 seconds ---
--- time for recursive 1 solution:  4.30777 seconds ---
--- time for recursive 2 solution:  4.14106 seconds ---
--- time for recursive 3 solution:  2.72964 seconds ---
--- time for better solution:  0.56211 seconds ---
```



In [13]:

```python
import sys
import time

import matplotlib.pyplot as pyplot


def time_func(name, limit, func):
    start_time = time.time()
    for n in range(1, limit):
        func(n)
    duration = (time.time() - start_time)
    print(f"--- time for {name} solution: %8.5f seconds ---" % duration)
    return duration


def fib1(n):
    # https://www.geeksforgeeks.org/fibonacci-series-in-python-using-for-loop/
    # Each appends the sum of the last two values of the list back to the list for n - 2 times
    # with the first two of the series being hard coded
    fib_series = [0, 1]
    for i in range(2, n):
        fib_series.append(fib_series[-1] + fib_series[-2])
    return fib_series


def fib2(n):
    # https://www.simplilearn.com/tutorials/python-tutorial/fibonacci-series
    # Each loop makes the next a value which will be added to the series a + b until the series has been
    # properly populated to length n
    fib_series = []
    a, b = 0, 1
    for i in range(n):
        fib_series.append(a)
        a, b = b, a + b
    return fib_series


def fib3(n):
    # https://www.simplilearn.com/tutorials/python-tutorial/fibonacci-series
    # Each loop makes the next a value which will be added to the series a + b until the series has been
    # properly populated to length n
    fib_series = []
    a, b = 0, 1
    while len(fib_series) < n:
        fib_series.append(a)
        a, b = b, a + b
    return fib_series


sys.setrecursionlimit(10 ** 6)


def fib4(s, n, a=0, b=1):
    # https://www.simplilearn.com/tutorials/python-tutorial/fibonacci-series
    # Recursively calls itself to generate a series with a being the sum of the last two values in the series
    # for a series of length n
    if n == 0:
        return a
    s.append(a)
    return fib4(s, n - 1, b, a + b)


normal = []
recursive1 = []
recursive2 = []
recursive3 = []
math_factorial = []

for limit in range(2000, 4000, 200):
    normal.append(time_func("for loop 1", limit, fib1))
    recursive1.append(time_func("for loop 2", limit, fib2))
    recursive2.append(time_func("while loop", limit, fib3))
    recursive3.append(time_func("recursive", limit, lambda n: fib4([], n)))

Y = list(range(2000, 4000, 200))

pyplot.plot(normal, Y, marker='o', label="For Loop 1", color='b')
pyplot.plot(recursive1, Y, marker='s', label="For Loop 2", color='g')
pyplot.plot(recursive2, Y, marker='^', label="While Loop", color='r')
pyplot.plot(recursive3, Y, marker='d', label="Recursive", color='m')

pyplot.xlabel("Duration of execution to calculate fibonacci series in seconds")
pyplot.ylabel("Fibonacci series length")
```

```python
pyplot.title("Time to calculate factorial using different functions")

pyplot.legend()

pyplot.show()

# The best function for this is the second for loop; this is likely because it stores
# values in a more time efficient space to access (locally rather than accessing list which adds
# overhead)
```

```
--- time for for loop 1 solution:  0.15640 seconds ---
--- time for for loop 2 solution:  0.12436 seconds ---
--- time for while loop solution:  0.21501 seconds ---
--- time for recursive solution:  0.35036 seconds ---
--- time for for loop 1 solution:  0.19590 seconds ---
--- time for for loop 2 solution:  0.23043 seconds ---
--- time for while loop solution:  0.19495 seconds ---
--- time for recursive solution:  0.51835 seconds ---
--- time for for loop 1 solution:  0.23814 seconds ---
--- time for for loop 2 solution:  0.17945 seconds ---
--- time for while loop solution:  0.23504 seconds ---
--- time for recursive solution:  0.54733 seconds ---
--- time for for loop 1 solution:  0.33534 seconds ---
--- time for for loop 2 solution:  0.21921 seconds ---
--- time for while loop solution:  0.27613 seconds ---
--- time for recursive solution:  0.59603 seconds ---
--- time for for loop 1 solution:  0.32103 seconds ---
--- time for for loop 2 solution:  0.24924 seconds ---
--- time for while loop solution:  0.32172 seconds ---
--- time for recursive solution:  0.69678 seconds ---
--- time for for loop 1 solution:  0.37554 seconds ---
--- time for for loop 2 solution:  0.29555 seconds ---
--- time for while loop solution:  0.37659 seconds ---
--- time for recursive solution:  0.87577 seconds ---
--- time for for loop 1 solution:  0.43425 seconds ---
--- time for for loop 2 solution:  0.33796 seconds ---
--- time for while loop solution:  0.43678 seconds ---
--- time for recursive solution:  0.91866 seconds ---
--- time for for loop 1 solution:  0.50857 seconds ---
--- time for for loop 2 solution:  0.39028 seconds ---
--- time for while loop solution:  0.49311 seconds ---
--- time for recursive solution:  1.04408 seconds ---
--- time for for loop 1 solution:  0.55932 seconds ---
--- time for for loop 2 solution:  0.43881 seconds ---
--- time for while loop solution:  0.55935 seconds ---
--- time for recursive solution:  1.17955 seconds ---
--- time for for loop 1 solution:  0.64120 seconds ---
--- time for for loop 2 solution:  0.49885 seconds ---
--- time for while loop solution:  0.64168 seconds ---
--- time for recursive solution:  1.31812 seconds ---
```