```
import math

import collections
triple = collections.namedtuple('triple', 'm fm simp')

def _quad_simpsons_mem(f: callable, a: float , fa: float, b: float, fb: float)->tup
    '''
        Evaluates Simpson's Rule, also returning m and f(m) to reuse.
    '''
    # Step1 in the book
    m = a + (b - a) / 2
    fm = f(m)
    simp = abs(b - a) / 6 * (fa + 4*fm + fb)
    return triple(m, fm, simp,)

def _quad_asr(f: callable, a: float, fa: float, b: float, fb: float, eps: float, wh
    '''
      Efficient recursive implementation of adaptive Simpson's rule.
      Function values at the start, middle, end of the intervals are retained.
    '''
    #Step 3-5 in the book
    lt = _quad_simpsons_mem(f, a, fa, m, fm)
    rt = _quad_simpsons_mem(f, m, fm, b, fb)
    delta = lt.simp + rt.simp - whole
    return (lt.simp + rt.simp + delta/10
        if (abs(delta) <= eps * 10) else
            _quad_asr(f, a, fa, m, fm, eps/2, lt.simp, lt.m, lt.fm) +
            _quad_asr(f, m, fm, b, fb, eps/2, rt.simp, rt.m, rt.fm)
    )

def quad_asr(f: callable, a: float, b: float, eps: float)->float:
    '''
        Integrate f from a to b using ASR with max error of eps.
    '''
    # Step2 call the recursive call
    fa = f(a)
    fb = f(b)
    t = _quad_simpsons_mem(f, a, fa, b, fb)
    return _quad_asr(f, a, fa, b, fb, eps, t.simp, t.m, t.fm)

 #Codes refers to: http://rosettacode.org/wiki/Numerical_integration/Adaptive_Simps


#5(a)
def f1(x):
  return math.exp(2*x) * math.sin(3*x)
(a, b,) = (1.0, 3.0,)
f_1 = quad_asr(f1, a, b, 1e-05)
print("Simpson's integration of f1(x) from {} to {} = {}\n".format(a, b, f_1))
```

```
Simpson's integration of f1(x) from 1.0 to 3.0 = 108.55528113347809
```

```
#5(c)
```

```
#9(c)
def f2(x):
  return 2*x*math.cos(2*x) - (x-2)*(x-2)
(a, b,) = (0.0, 5.0,)
f_2 = quad_asr(f2, a, b, 1e-05)
print("Simpson's integration of f2(x) from {} to {} = {}\n".format(a, b, f_2))
```

```
    Simpson's integration of f2(x) from 0.0 to 5.0 = -15.306307816864319
```

```
#9(a)
def f3(x):
  return math.sin(1/x)
(a, b,) = (0.1, 2.0,)
f_3 = quad_asr(f3, a, b, 1e-03)
print("Simpson's integration of f3(x) from {} to {} = {}\n".format(a, b, f_3))
```

```
    Simpson's integration of f3(x) from 0.1 to 2.0 = 1.1456032250446464
```

```
#9(b)
def f4(x):
  return math.cos(1/x)
(a, b,) = (0.1, 2.0,)
f_4 = quad_asr(f4, a, b, 1e-03)
print("Simpson's integration of f4(x) from {} to {} = {}\n".format(a, b, f_4))
```

```
    Simpson's integration of f4(x) from 0.1 to 2.0 = 0.673843376519166
```