Slither Package

This is a LaTeXpackage that makes it easy to include Python and Serpent codes in your documents with an aesthetically pleasing code block. It builds on a great package called pythonhighlight by Oliver. This is a huge upgrade over just using the verbatim package to include code.

https://github.com/olivierverdier/python-latex-highlighting/blob/master/pythonhighlight.sty

Python Codes

You can include code by typing it into the \begin{python}\end{python} environment.

Code 1: Hello!

```
    x=10
    print("Hello World") #comment
    try:
        a=2/x
    except ZeroDivisionError:
        print('undefined')
```

You can also use inline codes like import numpy or lambda x: x if x<=1 else fib(x-1) + fib(x-2) by using the \pyth{} control sequence.

Or you can save python files into your working directory and include them without the need to retype/copy-paste them using the \inputpython{<input.py>}{<firstline>}{<lastline> command. The line numbers are smart, so if <firstline> != 1, it will start from the appropriate line. Be careful with this, especially making sure not to break up multiline 'lines' like multiline comments or dictionaries.

Code 2: F strings

```
""""
This is a Multi-line Comment
Using Triple Quotes
"""

x = 4
print(f"The numeral four: {x}")
```

Serpent Input Files

It is a bit more complicated to use the Serpent capabilities. While I know all of the Control Words in Serpent, I do not know what you are going to name materials etc. You will have to open slither.sty, and add things that you would like to follow the 'Name' syntax highlighting file to the line under %Add Names here.

You can include input files by typing it into the \begin{serpent}\end{serpent} environment.

Code 3: Fuel!

```
/*
Enriched (4%) Uranium Metal
*/
mat fuel      -10.1
92235.03c     -0.04
92238.03c     -0.96
'string'
```

You can include individual commands inline like surf s1 sqc 0.0 0.0 100.0.

Or you can save serpent input files into your working directory and include them without the need to retype/copy-paste them using the \inputserpent{<input.txt>}{<firstline>}{<lastline> command.

Code 4: Physics Cards

```
% ______________________Physics cards_____
set pop 1000000 500 100 1
%set pop 20000 100 30 1
%set ngamma 1 %invokes production of prompt gammas in neutron reactions
set ures 1
set acelib "endfb71r1_p2" "endfb71r1" "jeff31u"
% Prints cross section data to [input]_xs0.m file.
set xsplot 100 1e-11 2.0 %comment out for temp sens
set power 0.4e6 % 400 KW thermal output
%
```

Wrapping Files over the Page Break

You'll likely have files long enough to wrap over multiple pages. This is simple to deal with using the input¡python/serpent¿ commands, as these natively allow you to handle page numbers. It's a little hacky as part of the code lays outside of the code-float, but it still allows you to reference it using the label.

Code 5: Python OOP Helium Cycle

```python
import convert

class Fluid:
    R = 8.3145 #kJ/kmol-K
    cPs = {'helium': 5.1926,
           'argon': 0.5203,
           'neon': 1.0299,
           'air':1.005,
           'hydrogen':14.307}
    MWs = {'helium': 4.003,
           'argon': 39.948,
           'neon': 20.183,
           'air':28.97,
           'hydrogen':2.016}

    def __init__(self,gas):
        self.cP = Fluid.cPs[gas] #kJ/kg-K
        self.MW = Fluid.MWs[gas]
        self.specR = self.R/self.MW #kJ/kg-K
        self.cV = self.cP-self.specR #kJ/kg-K
        self.gamma = self.cP/self.cV
        self.gamma_bar = (self.gamma-1)/self.gamma


############################################################
class State:
    MASSFLOW = 0 #kg/s

    states = {1:'Compressor Inlet',
              2:'Compressor Outlet',
              3:'Expander Inlet',
              4:'Expander Outlet'}

    lowP,highP = 0.93,7
    temperatures = {1:25,
                    2:'???',
                    3:900,
                    4:'???'} #degC
```

```
    pressures = {1:lowP,
                 2:highP,
                 3:highP,
                 4:lowP} #MPa

    list = []

    def __init__(self, stream):
        self.stream = stream
        self.name = State.states[stream]
        self.pressure = State.pressures[stream]
        self.temperature = State.temperatures[stream]
        State.list.append(self)


    def get_info(self):
        print(f'Stream {self.stream} is the {self.name}. T = {round(self.temperature,1)} degC, P= {self.pressure} MPa ')

    @classmethod
    def carnot_efficiency(cls):
        hotT, coldT = max(stream.temperature for stream in cls.list), min(stream.temperature for stream in cls.list)
        hotT, coldT = convert.Temperature(hotT,'C','K'), convert.Temperature(coldT,'C','K')
        eta = 100*(1-coldT/hotT)
        eta = round(eta,2)
        print(f'Carnot Efficiency: {eta}%')

    @classmethod
    def massflow(cls):
        print(f'Mass Flow Rate {round(cls.MASSFLOW,1)} kg/s')

##################################
class Equipment:
    list=[]

    def __init__(self, inlet, outlet):
        self.inlet = inlet
        self.outlet = outlet
        Equipment.list.append(self)

class Work(Equipment):
    POWER = 0

    efficiencies = {'compressor': 0.9,
                    'expander': 0.9}

    eff_exponent = {'compressor': -1,
                    'expander': 1}
```

```
    @classmethod
    def net_power(cls):
        net = round(Heat.POWER_NET,1)
        print(f'Net Shaft Power: {net} MW' )

    def __init__(self, inlet, outlet, descr):
        super().__init__(inlet, outlet)
        self.descr = descr
        self.work = '???'

    def shaft_work(self,gas):
        inT = convert.Temperature(self.inlet.temperature,'C','K') #K
        outTrev = inT*(self.outlet.pressure/self.inlet.pressure)**gas.gamma_bar #K
        self.work = gas.cP*(inT-outTrev)*Work.efficiencies[self.descr]**Work.eff_exponent[self.descr]
        return self.work

    def outT(self, gas):
        return self.inlet.temperature - self.shaft_work(gas)/gas.cP

    def find_power(self):
        self.power = self.work*State.MASSFLOW
        self.power = convert.Metric(self.power, 'k', 'M')
        Work.POWER +=self.power
        self.power = round(self.power,1)

    def get_info(self):
        print(f'{self.descr}: {self.power} MW')

class Heat(Equipment):
    powers = {'reactor': 600,
              'cooler': '???'}

    POWER_IN = POWER_NET = powers['reactor']

    def __init__(self, inlet, outlet, descr):
        super().__init__(inlet, outlet)
        self.descr = descr
        self.power = Heat.powers[descr]


    def find_massflow(self,gas):
        State.MASSFLOW = convert.Metric(self.power,'M','k')/(gas.cP*(self.outlet.temperature-self.inlet.temperature))
```

```python

    def find_power(self,gas):
        self.power = State.MASSFLOW*gas.cP*(self.outlet.temperature
                                            -self.inlet.temperature)
        self.power = convert.Metric(self.power,'k','M')
        Heat.POWER_NET+=self.power
        self.power = round(self.power,1)

    def get_info(self):
        print (f'{self.descr}: {self.power} MW')

    @classmethod
    def thermal_efficiency(cls):
        eta = cls.POWER_NET/cls.POWER_IN*100
        eta = round(eta,2)
        print(f'Thermal Efficiency: {eta}%')
############################################################
def main():
    #Initialize Fluids
    helium = Fluid('helium')
    argon  = Fluid('argon')
    neon = Fluid('neon')
    air = Fluid('air')
    hydrogen = Fluid('hydrogen')
    GAS = hydrogen

    #Initialize Streams
    compressorIn = State(1)
    compressorOut = State(2)
    expanderIn = State(3)
    expanderOut = State(4)

    #Initialize Equipment
    compressor = Work(compressorIn,compressorOut,'compressor')
    expander = Work(expanderIn,expanderOut,'expander')
    reactor = Heat(compressorOut,expanderIn,'reactor')
    cooler = Heat(expanderOut,compressorIn,'cooler')

    #Solve Unknown States
    compressorOut.temperature = compressor.outT(GAS)
    expanderOut.temperature = expander.outT(GAS)

    #Solve Powers
    reactor.find_massflow(GAS)
    cooler.find_power(GAS)
    compressor.find_power()
    expander.find_power()

def print_out():
```

```
    for stream in State.list:
        stream.get_info()

    State.massflow()

    for equipment in Equipment.list:
        equipment.get_info()

    Work.net_power()

    Heat.thermal_efficiency()
    State.carnot_efficiency()


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