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, or the syntax highlighting may fail.

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

```
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}
```

```
87
88
89      @classmethod
90      def net_power(cls):
91          net = round(Heat.POWER_NET,1)
92          print(f'Net Shaft Power: {net} MW' )
93
94      def __init__(self, inlet, outlet, descr):
95          super().__init__(inlet, outlet)
96          self.descr = descr
97          self.work = '???'
98
99      def shaft_work(self,gas):
100         inT = convert.Temperature(self.inlet.temperature,'C','K') #
                                                K
101         outTrev = inT*(self.outlet.pressure/self.inlet.pressure)**
                                                gas.gamma_bar #K
102         self.work = gas.cP*(inT-outTrev)*Work.efficiencies[self.
                                                descr]**Work.eff_exponent
                                                [self.descr]
103         return self.work
104
105     def outT(self, gas):
106         return self.inlet.temperature - self.shaft_work(gas)/gas.cP
107
108     def find_power(self):
109         self.power = self.work*State.MASSFLOW
110         self.power = convert.Metric(self.power, 'k', 'M')
111         Work.POWER +=self.power
112         self.power = round(self.power,1)
113
114     def get_info(self):
115         print(f'{self.descr}: {self.power} MW')
116
117 class Heat(Equipment):
118     powers = {'reactor': 600,
119               'cooler': '???'}
120
121     POWER_IN = POWER_NET = powers['reactor']
122
123     def __init__(self, inlet, outlet, descr):
124         super().__init__(inlet, outlet)
125         self.descr = descr
126         self.power = Heat.powers[descr]
127
128
129     def find_massflow(self,gas):
130         State.MASSFLOW = convert.Metric(self.power,'M','k')/(gas.cP
                                                *(self.outlet.temperature
                                                -self.inlet.temperature))
```

```
131
132     def find_power(self,gas):
133         self.power = State.MASSFLOW*gas.cP*(self.outlet.temperature
                                                -self.inlet.temperature)
134         self.power = convert.Metric(self.power,'k','M')
135         Heat.POWER_NET+=self.power
136         self.power = round(self.power,1)
137
138     def get_info(self):
139         print (f'{self.descr}: {self.power} MW')
140
141     @classmethod
142     def thermal_efficiency(cls):
143         eta = cls.POWER_NET/cls.POWER_IN*100
144         eta = round(eta,2)
145         print(f'Thermal Efficiency: {eta}%')
146 ############################################################
147 def main():
148     #Initialize Fluids
149     helium = Fluid('helium')
150     argon  = Fluid('argon')
151     neon = Fluid('neon')
152     air = Fluid('air')
153     hydrogen = Fluid('hydrogen')
154     GAS = hydrogen
155
156     #Initialize Streams
157     compressorIn = State(1)
158     compressorOut = State(2)
159     expanderIn = State(3)
160     expanderOut = State(4)
161
162     #Initialize Equipment
163     compressor = Work(compressorIn,compressorOut,'compressor')
164     expander = Work(expanderIn,expanderOut,'expander')
165     reactor = Heat(compressorOut,expanderIn,'reactor')
166     cooler = Heat(expanderOut,compressorIn,'cooler')
167
168     #Solve Unknown States
169     compressorOut.temperature = compressor.outT(GAS)
170     expanderOut.temperature = expander.outT(GAS)
171
172     #Solve Powers
173     reactor.find_massflow(GAS)
174     cooler.find_power(GAS)
175     compressor.find_power()
176     expander.find_power()
177
178 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()
```