Print Formatting

3 Methods

1. Modulo (%) method
2. .format() method
3. f-string method

Challenge is different data types have different issues

(str, float, int)

Modulus Method

- Old method and may be in packages you import
- Types
  - %s - represents string
  - %i - represents integer
  - %d - represents decimal integer
  - %f - represents float

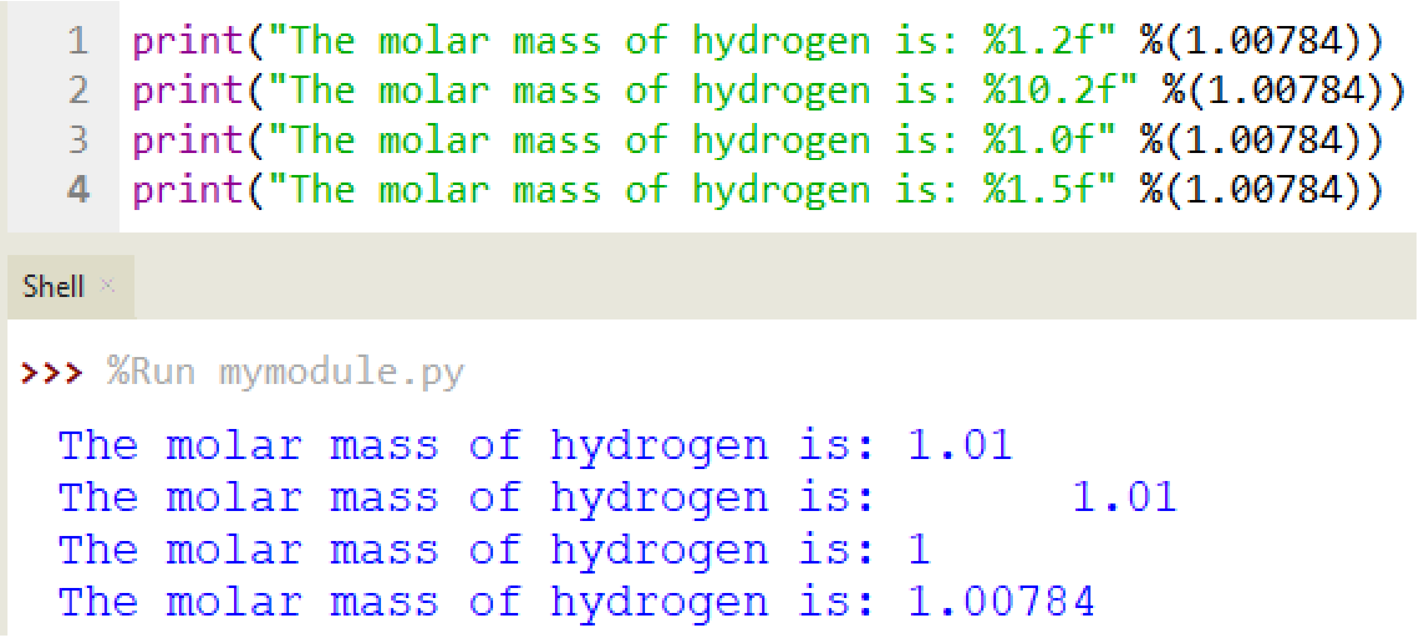
%[<width>][.<precision>]<type>

Modulus Method

%[<width>][.<precision>]<type>

%6.2f (f,d or i)

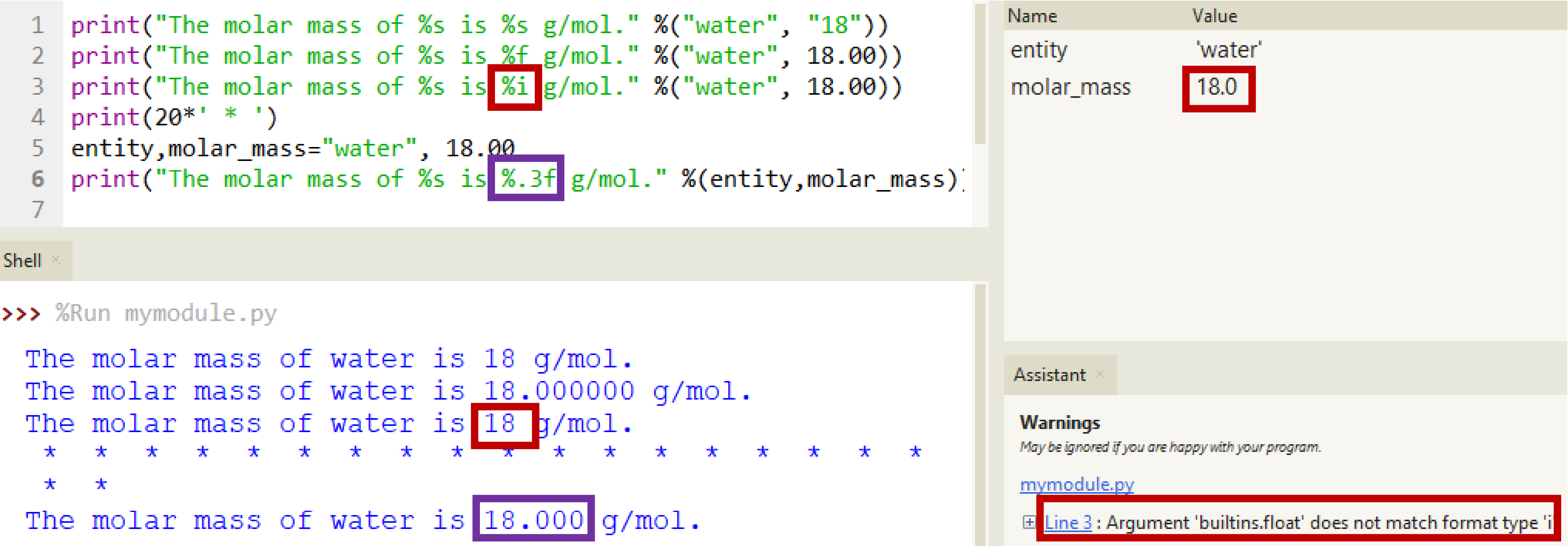
$$\underbrace{\#\#\#.\underbrace{\#\#}_{2}}_{6}$$



```
print("The molar mass of hydrogen is: %1.2f" %(1.00784))
print("The molar mass of hydrogen is: %10.2f" %(1.00784))
print("The molar mass of hydrogen is: %1.0f" %(1.00784))
print("The molar mass of hydrogen is: %1.5f" %(1.00784))
```

```
>>> %Run mymodule.py

 The molar mass of hydrogen is: 1.01
 The molar mass of hydrogen is:       1.01
 The molar mass of hydrogen is: 1
 The molar mass of hydrogen is: 1.00784
```

Modulus Method



```
1  print("The molar mass of %s is %s g/mol." %("water", "18"))
2  print("The molar mass of %s is %f g/mol." %("water", 18.00))
3  print("The molar mass of %s is %i g/mol." %("water", 18.00))
4  print(20*' * ')
5  entity,molar_mass="water", 18.00
6  print("The molar mass of %s is %.3f g/mol." %(entity,molar_mass))
7
```

```
>>> %Run mymodule.py

 The molar mass of water is 18 g/mol.
 The molar mass of water is 18.000000 g/mol.
 The molar mass of water is 18 g/mol.
  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
  *  *
 The molar mass of water is 18.000 g/mol.
```

Name	Value
entity	'water'
molar_mass	18.0

Warnings

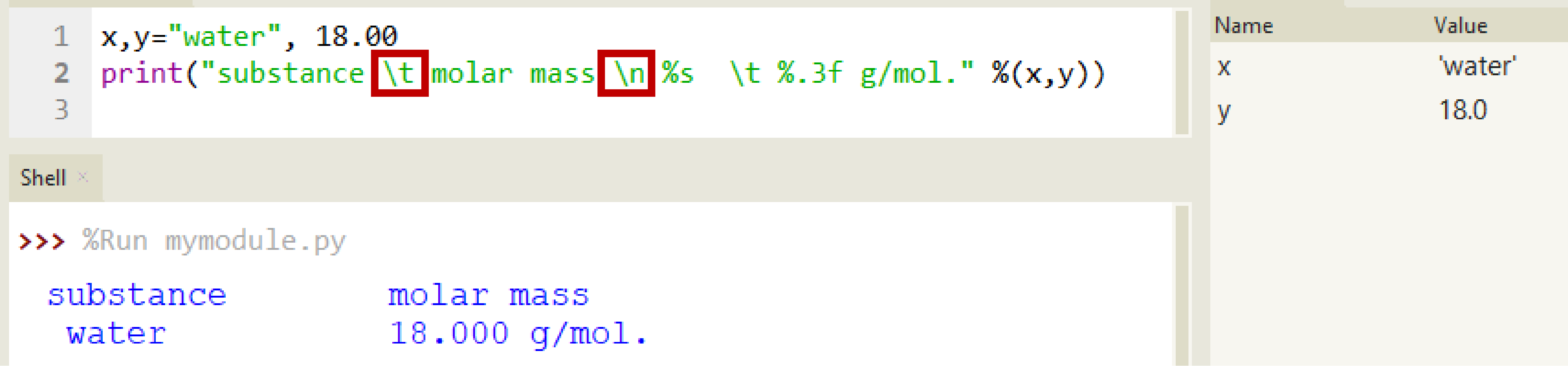
May be ignored if you are happy with your program.

mymodule.py

⊞ Line 3 : Argument 'builtins.float' does not match format type 'i

Modulus Method

\t - tab
\n - new line



```
x,y="water", 18.00
print("substance \t molar mass \n %s  \t %.3f g/mol." %(x,y))

```

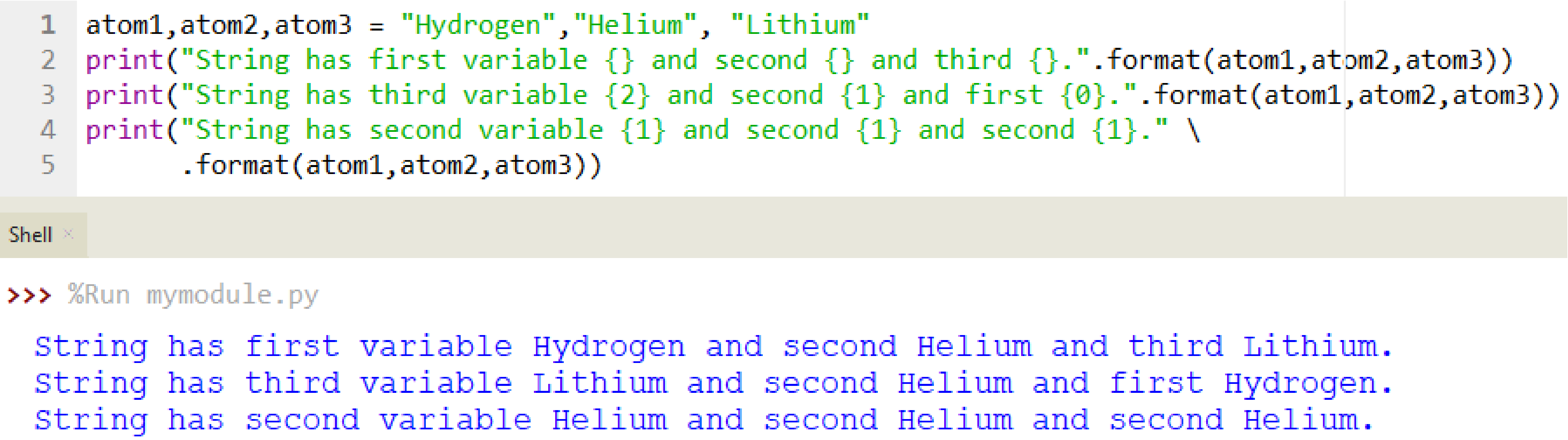
```
>>> %Run mymodule.py

 substance       molar mass
  water          18.000 g/mol.
```

Name	Value
x	'water'
y	18.0

.format method

"text {} text {} text {}.".format(v1,v2,v3)

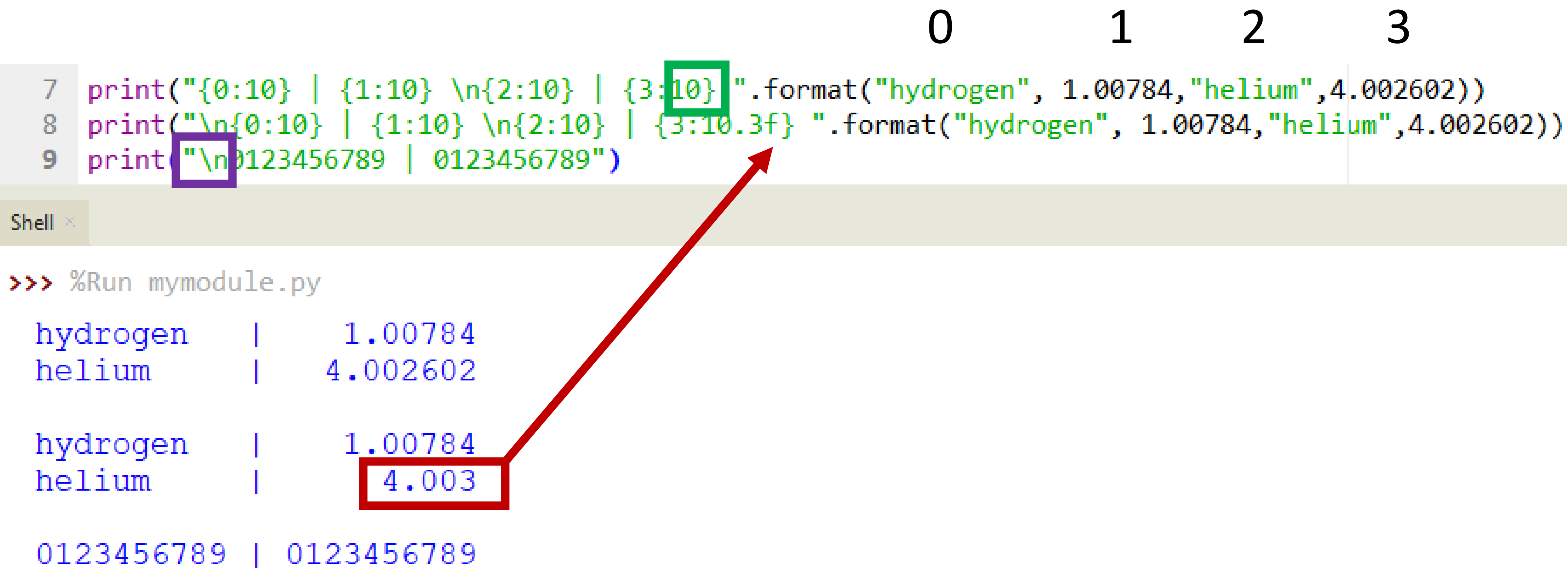
- can use index numbers
- place width and precision in {}



```python
atom1,atom2,atom3 = "Hydrogen","Helium", "Lithium"
print("String has first variable {} and second {} and third {}.".format(atom1,atom2,atom3))
print("String has third variable {2} and second {1} and first {0}.".format(atom1,atom2,atom3))
print("String has second variable {1} and second {1} and second {1}." \
      .format(atom1,atom2,atom3))
```

```
>>> %Run mymodule.py

 String has first variable Hydrogen and second Helium and third Lithium.
 String has third variable Lithium and second Helium and first Hydrogen.
 String has second variable Helium and second Helium and second Helium.
```

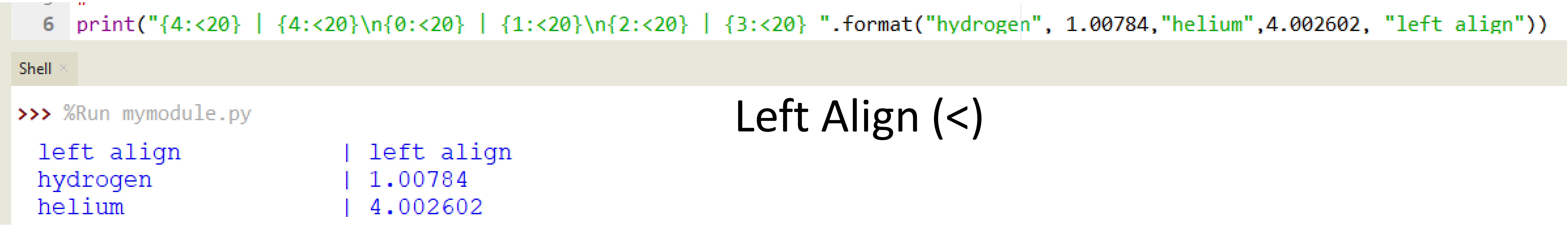
.format method



```
print("{0:10} | {1:10} \n{2:10} | {3:10} ".format("hydrogen", 1.00784,"helium",4.002602))
print("\n{0:10} | {1:10} \n{2:10} | {3:10.3f} ".format("hydrogen", 1.00784,"helium",4.002602))
print("\n0123456789 | 0123456789")
```

```
>>> %Run mymodule.py
hydrogen   |    1.00784
helium     |   4.002602

hydrogen   |    1.00784
helium     |      4.003

0123456789 | 0123456789
```

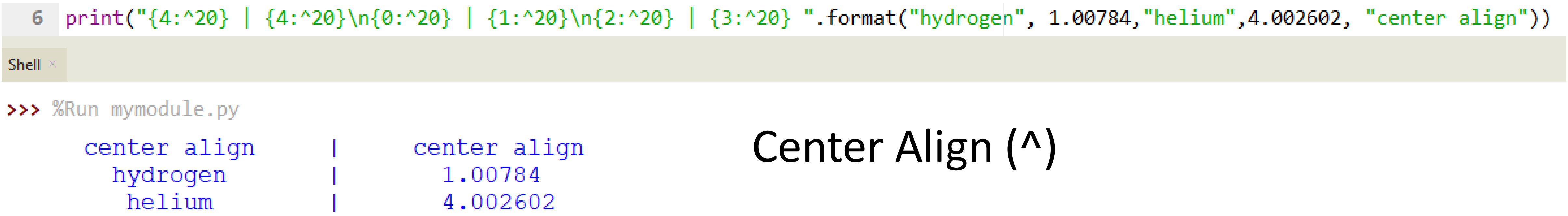
.format method



```
print("{4:<20} | {4:<20}\n{0:<20} | {1:<20}\n{2:<20} | {3:<20} ".format("hydrogen", 1.00784,"helium",4.002602, "left align"))
```

```
>>> %Run mymodule.py
left align           | left align
hydrogen             | 1.00784
helium               | 4.002602
```

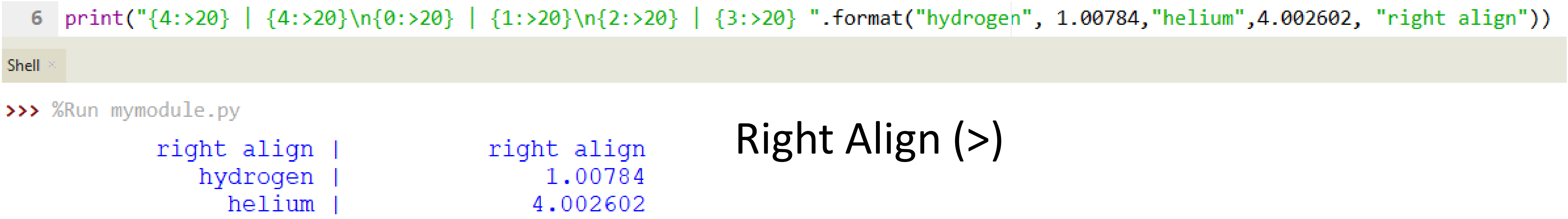
Left Align (<)



```
print("{4:^20} | {4:^20}\n{0:^20} | {1:^20}\n{2:^20} | {3:^20} ".format("hydrogen", 1.00784,"helium",4.002602, "center align"))
```

```
>>> %Run mymodule.py
    center align     |     center align
      hydrogen       |       1.00784
       helium        |      4.002602
```

Center Align (^)



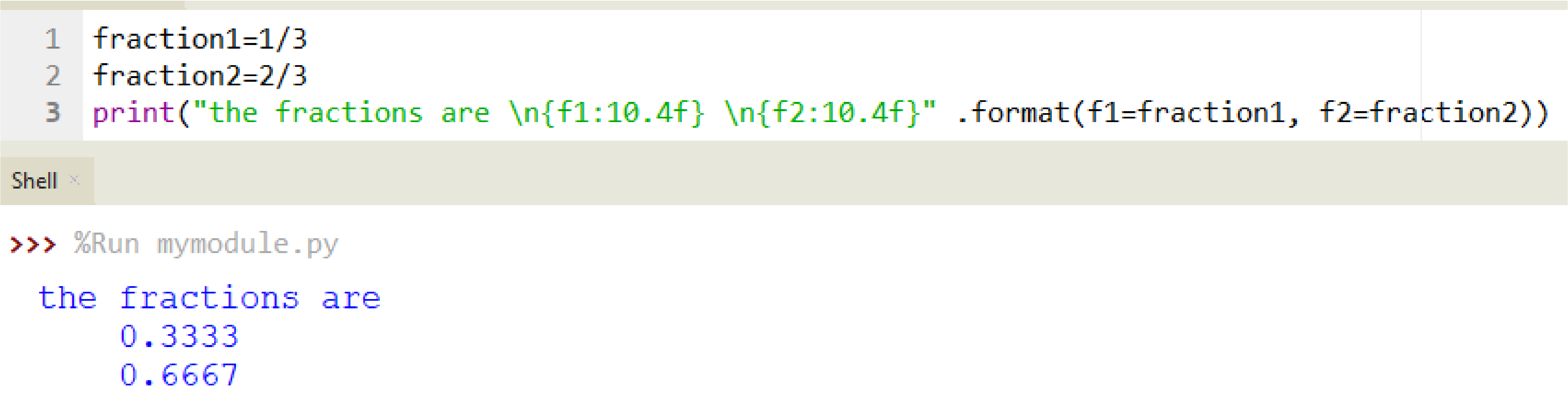
```
print("{4:>20} | {4:>20}\n{0:>20} | {1:>20}\n{2:>20} | {3:>20} ".format("hydrogen", 1.00784,"helium",4.002602, "right align"))
```

```
>>> %Run mymodule.py
         right align |          right align
            hydrogen |              1.00784
              helium |             4.002602
```

Right Align (>)

.format method

Using variable names in format statements

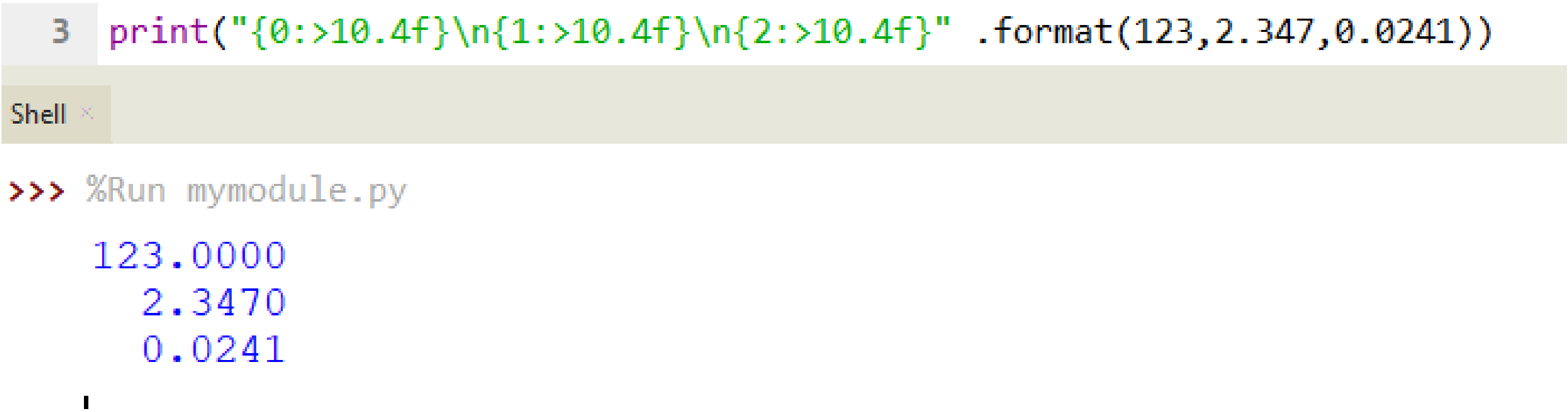


```python
fraction1=1/3
fraction2=2/3
print("the fractions are \n{f1:10.4f} \n{f2:10.4f}" .format(f1=fraction1, f2=fraction2))
```

```
>>> %Run mymodule.py

 the fractions are
     0.3333
     0.6667
```

.format method

Align by decimal



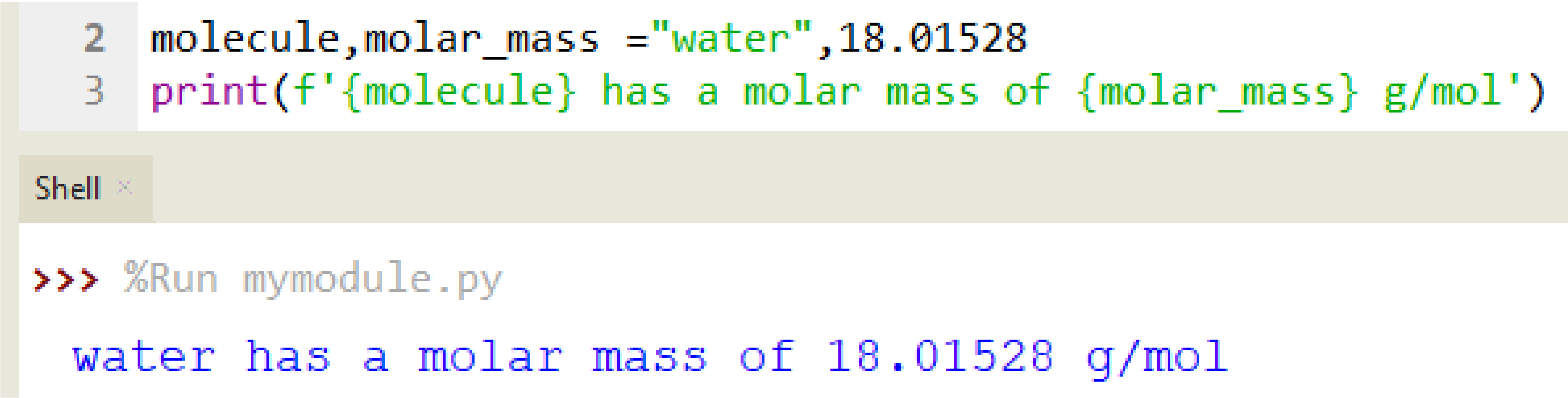
```
print("{0:>10.4f}\n{1:>10.4f}\n{2:>10.4f}" .format(123,2.347,0.0241))
```

```
>>> %Run mymodule.py

  123.0000
    2.3470
    0.0241
```

- Right align
- Choose width that covers digits from largest to smallest number
- Choose precision that covers most precise number

f strings

- Introduced in Python 3.6
- place f in front of string
- allows you to skip .format step



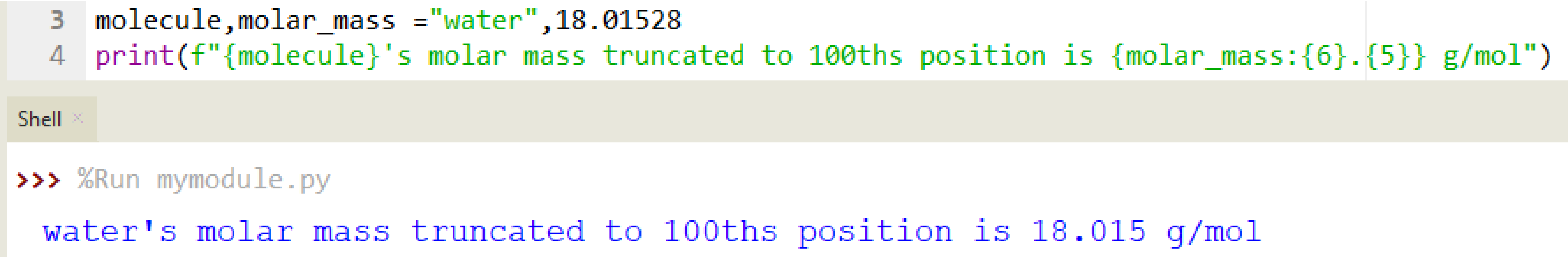
```python
molecule,molar_mass ="water",18.01528
print(f'{molecule} has a molar mass of {molar_mass} g/mol')
```

```
>>> %Run mymodule.py

  water has a molar mass of 18.01528 g/mol
```

f strings

- Precision is based on total digits (not right of decimal)

{molar_mass:{width}.{precision}}

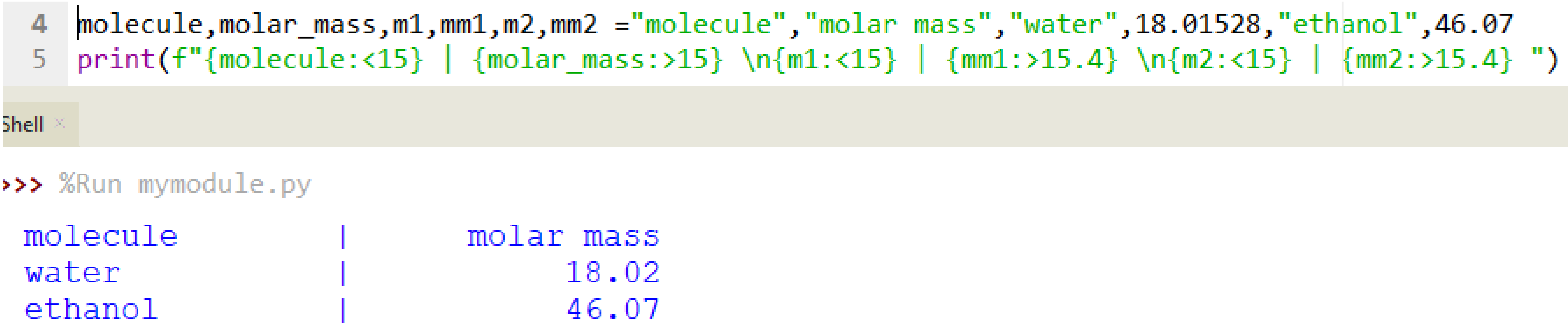


```
molecule,molar_mass ="water",18.01528
print(f"{molecule}'s molar mass truncated to 100ths position is {molar_mass:{6}.{5}} g/mol")
```

```
>>> %Run mymodule.py
  water's molar mass truncated to 100ths position is 18.015 g/mol
```

f strings

- Aligning Decimal Point



```
molecule,molar_mass,m1,mm1,m2,mm2 ="molecule","molar mass","water",18.01528,"ethanol",46.07
print(f"{molecule:<15} | {molar_mass:>15} \n{m1:<15} | {mm1:>15.4} \n{m2:<15} | {mm2:>15.4} ")
```

```
>>> %Run mymodule.py

 molecule        |      molar mass
 water           |           18.02
 ethanol         |           46.07
```

Logic & Control Structures

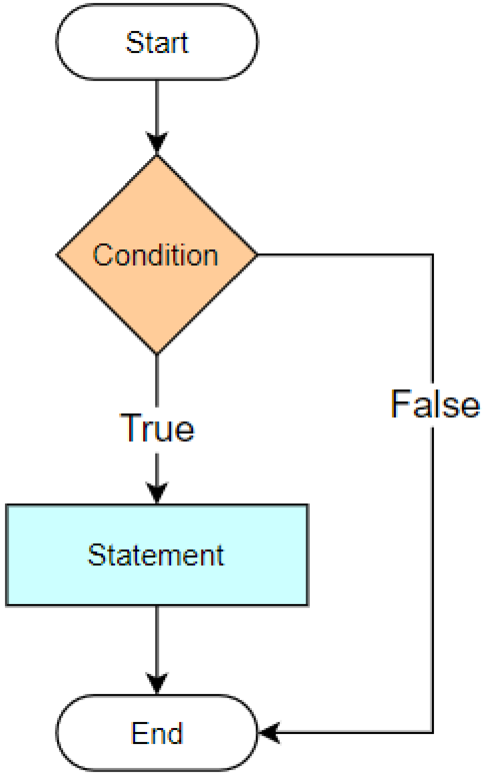
- Control structures allow program to use logic to execute code
- Control syntax uses colons and indentation to define blocks

Two basic types

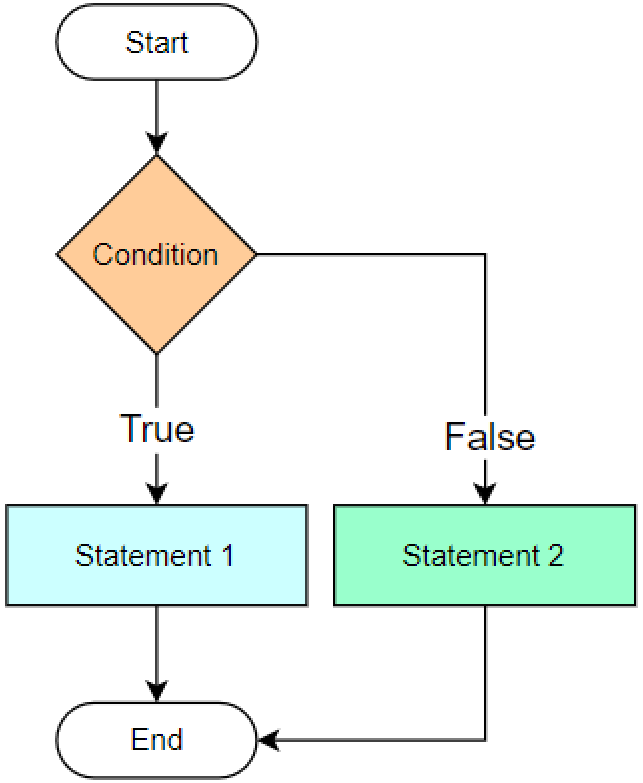
1. Conditionals - execute one or more statements if a condition is met

2. Loops - iterate through a statement based on a condition

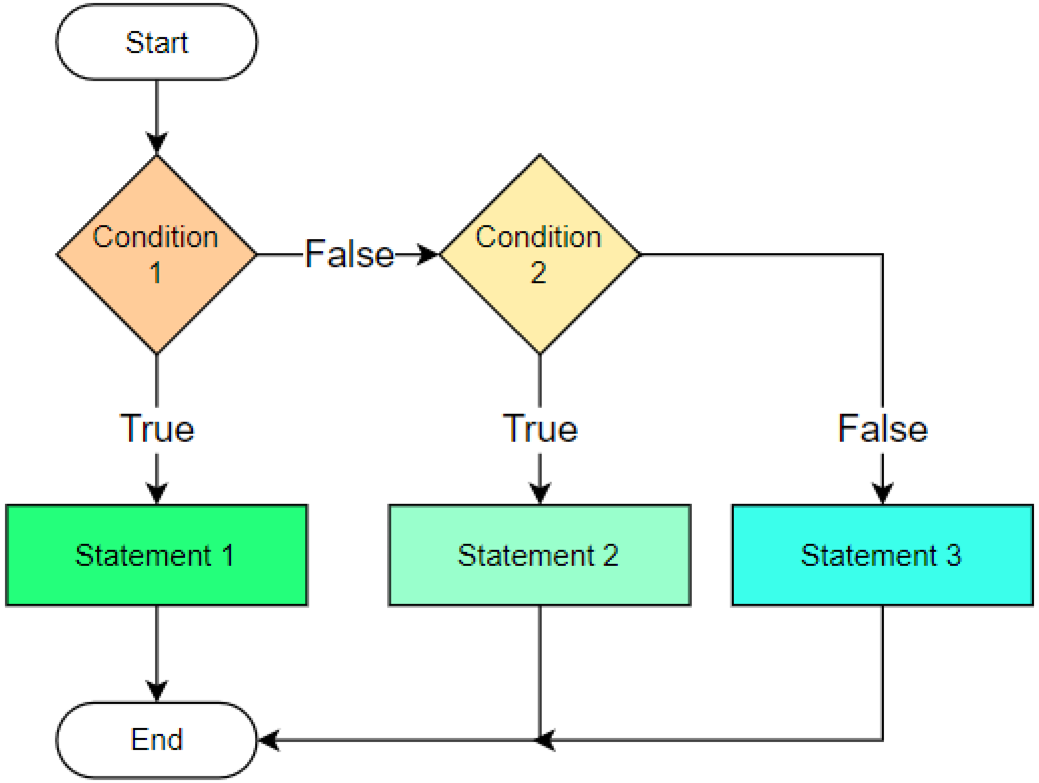
Conditional Logic Structures



1. If Statement



2. If-Else Statement

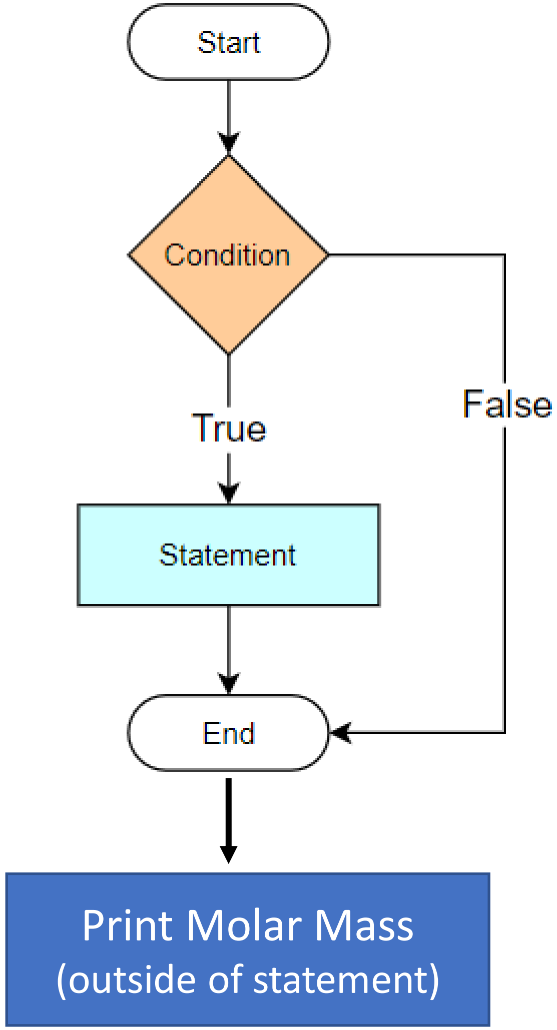


3. If-Elif-Else Statement

if Statement

Note syntax features:

- colon
- indentation



Start

Condition

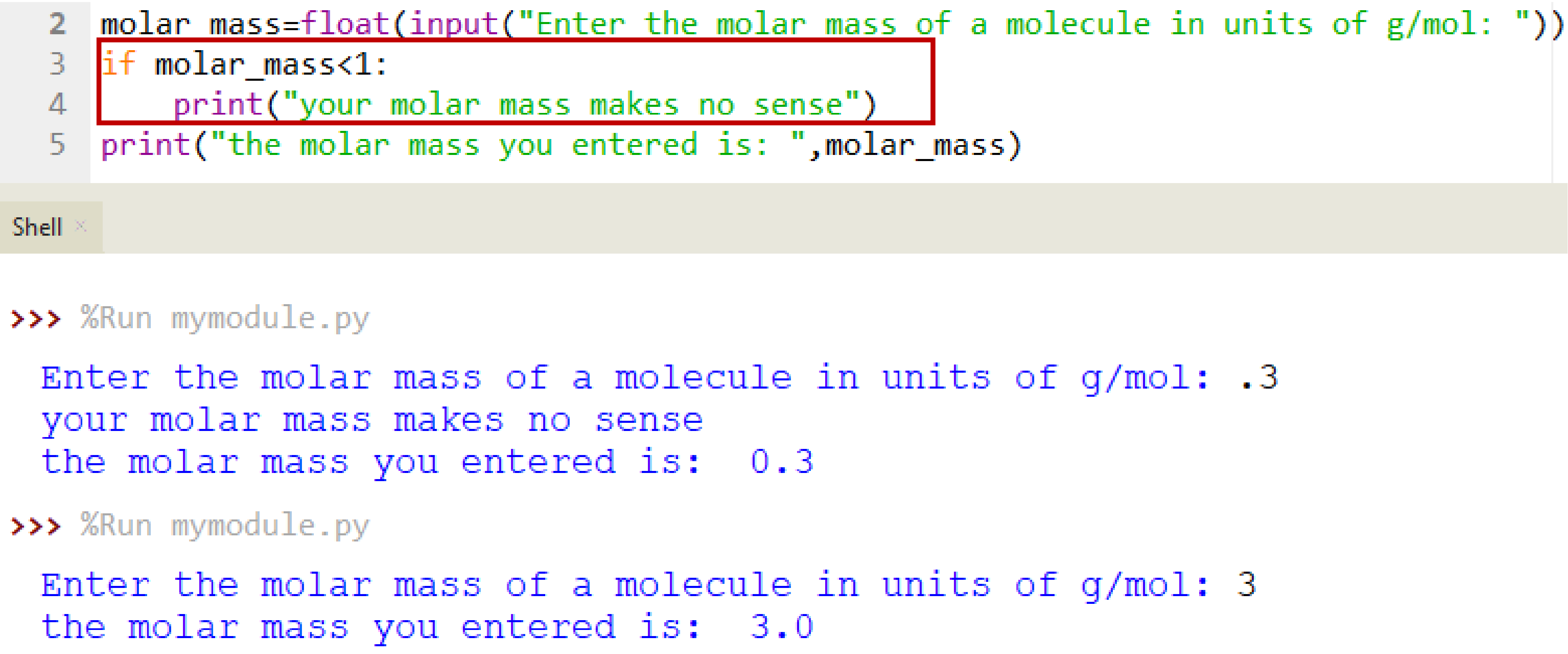
True

False

Statement

End

Print Molar Mass (outside of statement)

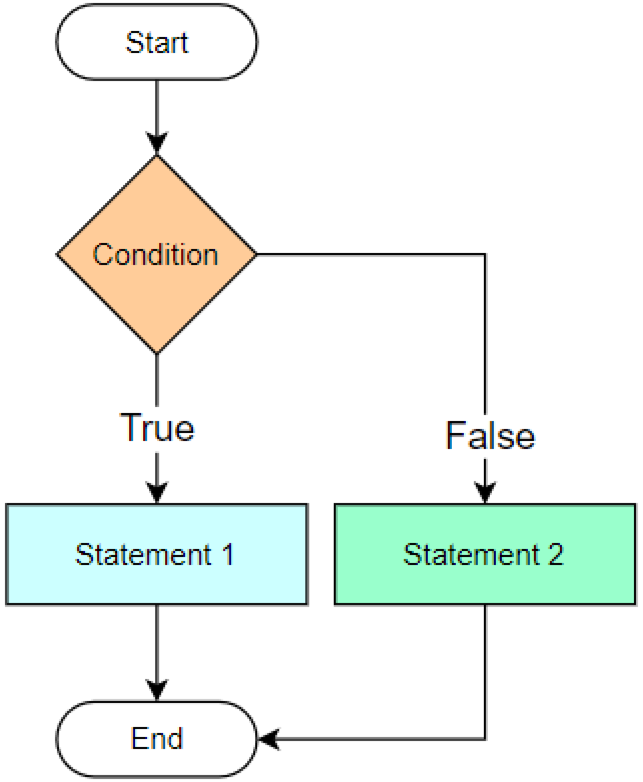


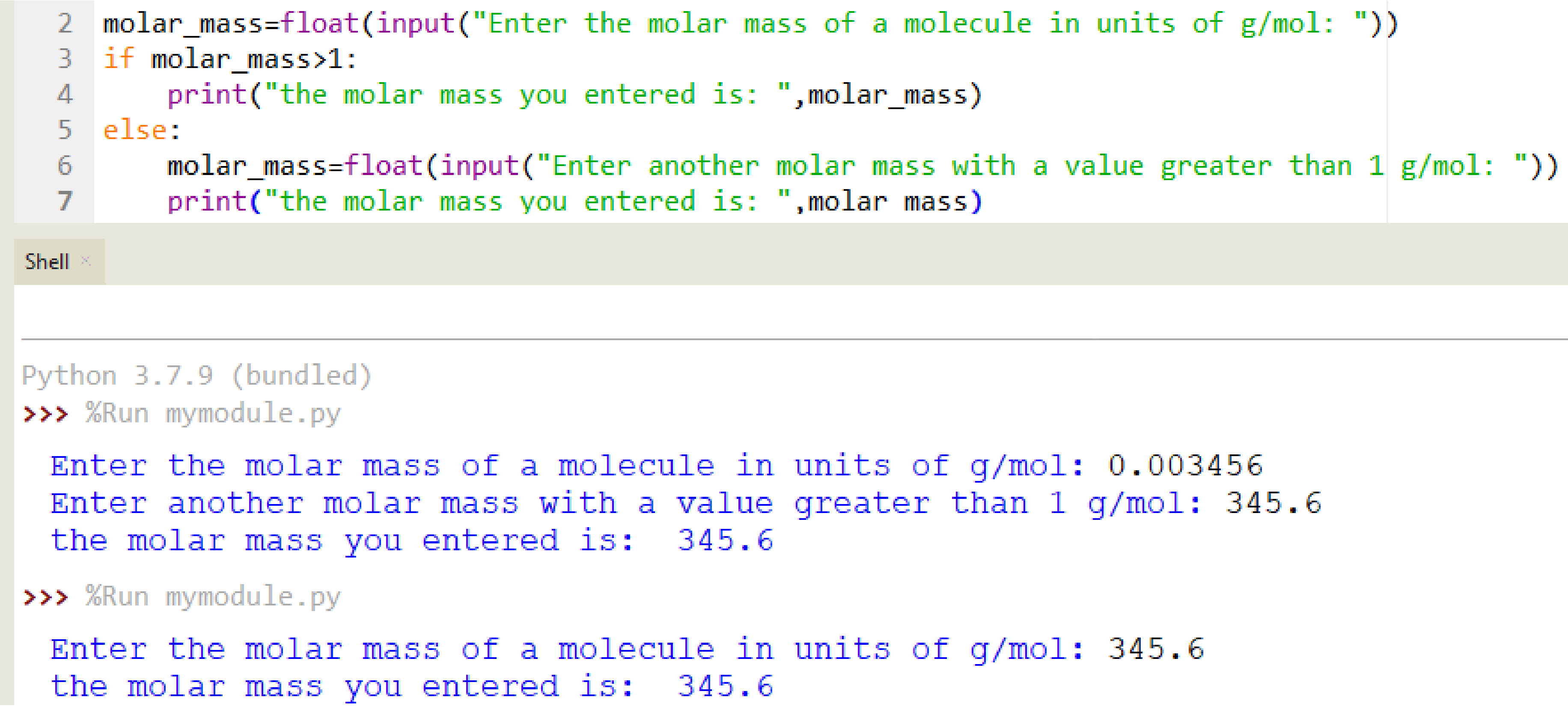
```
molar_mass=float(input("Enter the molar mass of a molecule in units of g/mol: "))
if molar_mass<1:
    print("your molar mass makes no sense")
print("the molar mass you entered is: ",molar_mass)
```

Shell

```
>>> %Run mymodule.py

 Enter the molar mass of a molecule in units of g/mol: .3
 your molar mass makes no sense
 the molar mass you entered is:  0.3

>>> %Run mymodule.py

 Enter the molar mass of a molecule in units of g/mol: 3
 the molar mass you entered is:  3.0
```

if-else Statement

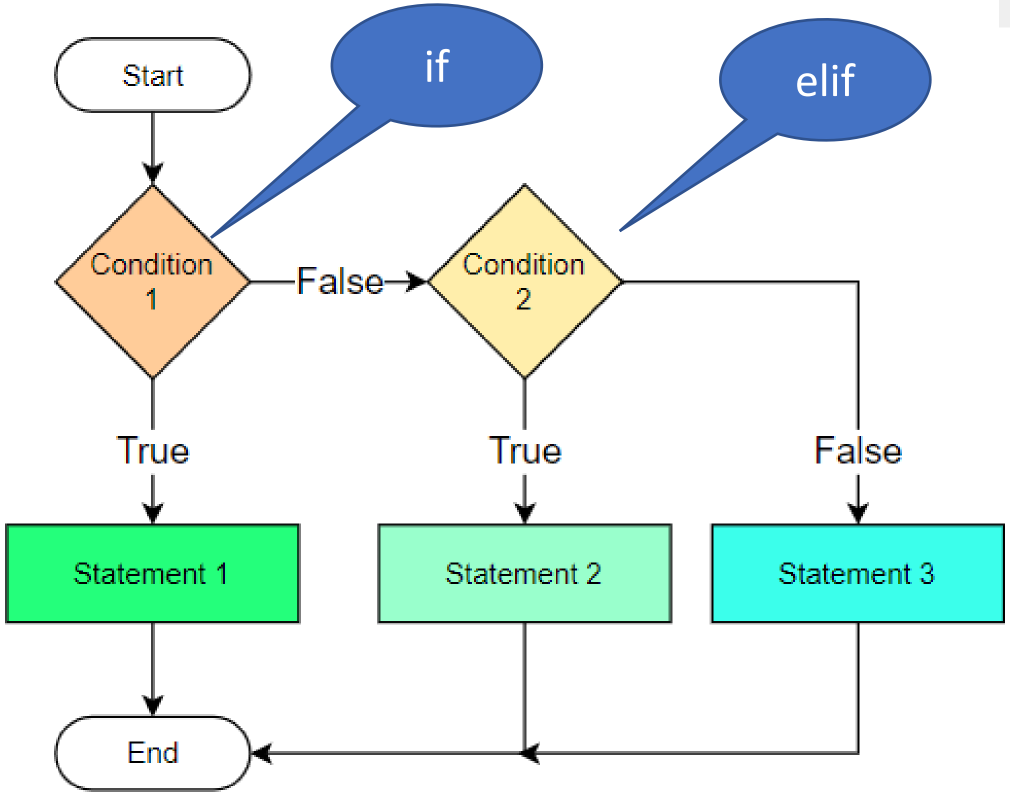


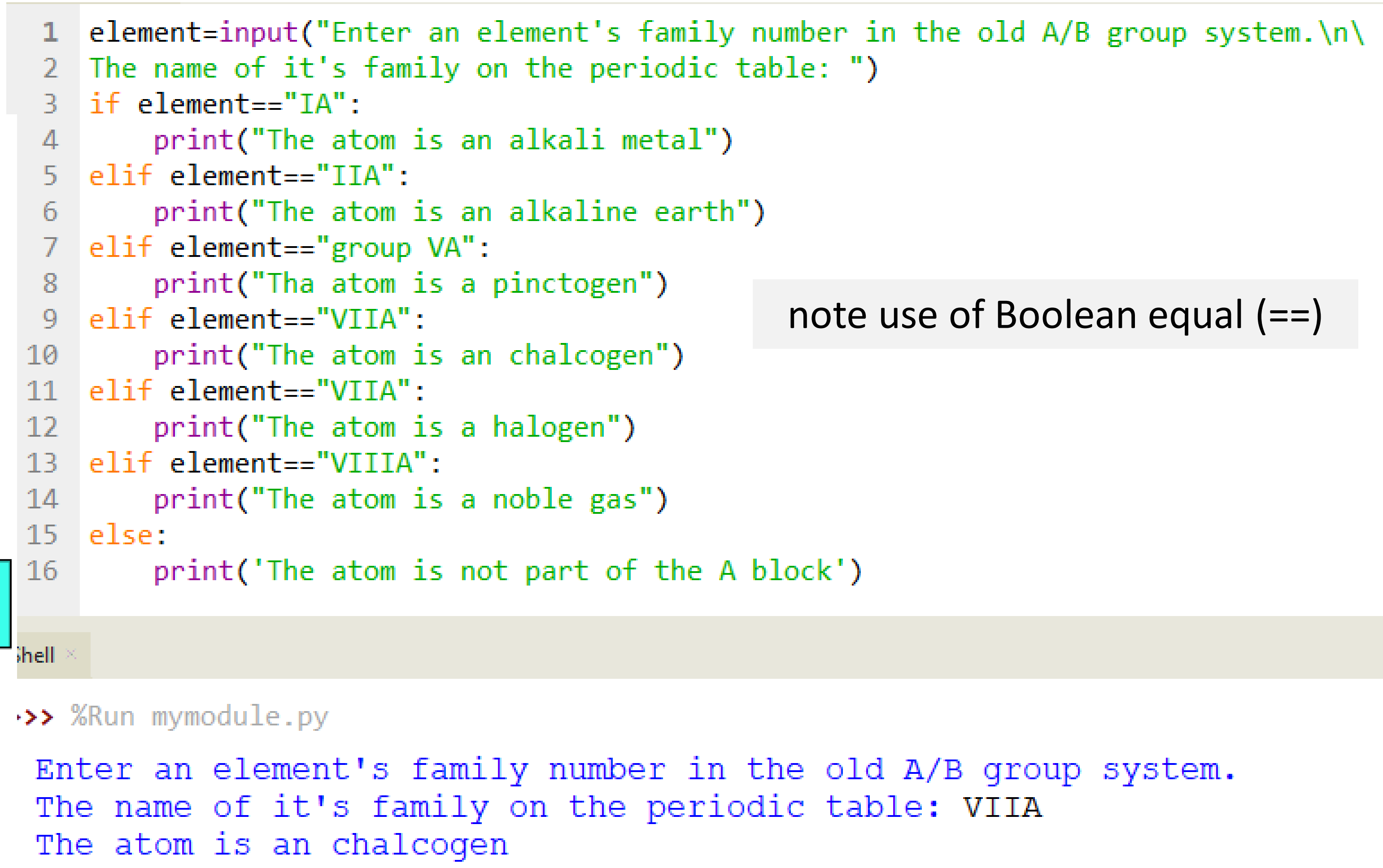


```
molar_mass=float(input("Enter the molar mass of a molecule in units of g/mol: "))
if molar_mass>1:
    print("the molar mass you entered is: ",molar_mass)
else:
    molar_mass=float(input("Enter another molar mass with a value greater than 1 g/mol: "))
    print("the molar mass you entered is: ",molar_mass)
```

```
Python 3.7.9 (bundled)
>>> %Run mymodule.py

 Enter the molar mass of a molecule in units of g/mol: 0.003456
 Enter another molar mass with a value greater than 1 g/mol: 345.6
 the molar mass you entered is:  345.6

>>> %Run mymodule.py

 Enter the molar mass of a molecule in units of g/mol: 345.6
 the molar mass you entered is:  345.6
```

if-elif-else statement





```python
element=input("Enter an element's family number in the old A/B group system.\n\
The name of it's family on the periodic table: ")
if element=="IA":
    print("The atom is an alkali metal")
elif element=="IIA":
    print("The atom is an alkaline earth")
elif element=="group VA":
    print("Tha atom is a pinctogen")
elif element=="VIIA":
    print("The atom is an chalcogen")
elif element=="VIIA":
    print("The atom is a halogen")
elif element=="VIIIA":
    print("The atom is a noble gas")
else:
    print('The atom is not part of the A block')
```

```
>> %Run mymodule.py

 Enter an element's family number in the old A/B group system.
 The name of it's family on the periodic table: VIIA
 The atom is an chalcogen
```

note use of Boolean equal (==)

Loop Structures

- Involve Iterations
  - Iterations may be items of an object like the content of a list
  - Iterations may be on the result of a Boolean logic statement

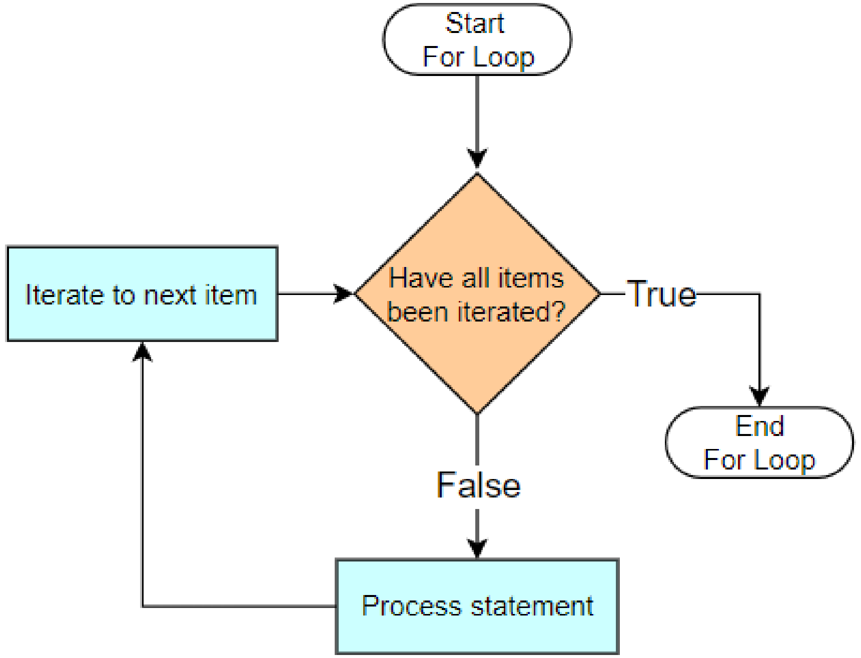
3 Loop Structures

1. For Loops
2. While Loops
3. Nested Loops

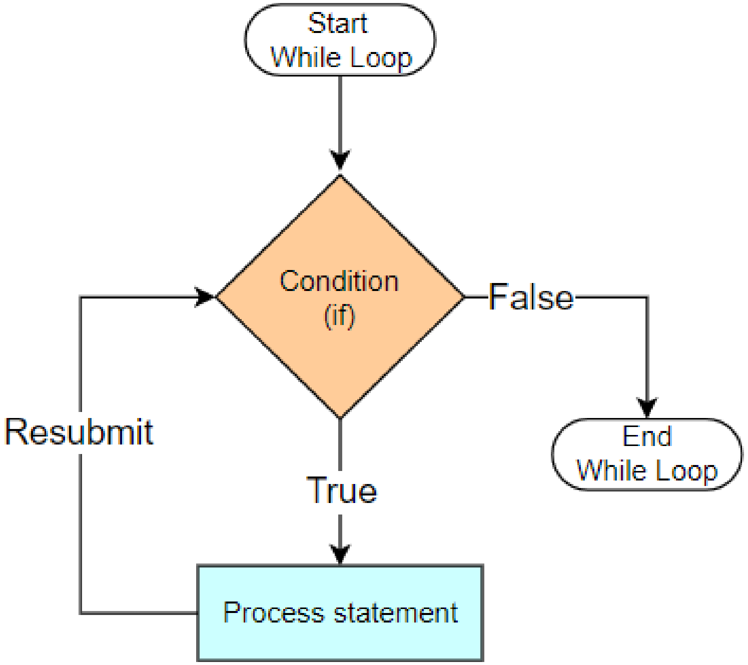
Loop Control Statements

1. break statement
2. pass statement
3. continue statement

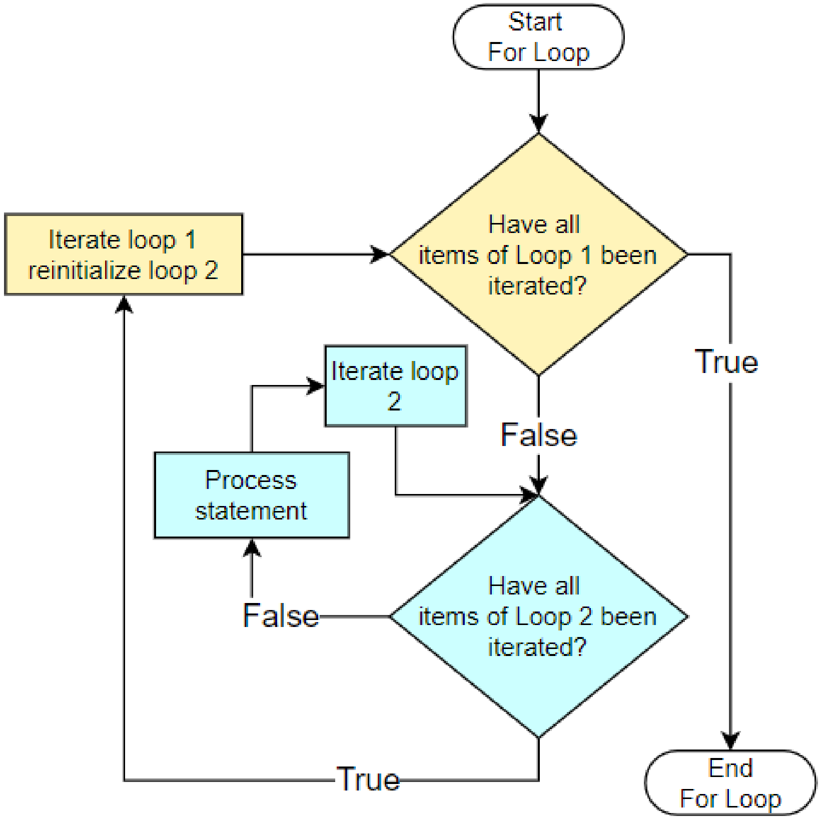
Loop Structures



For Loop

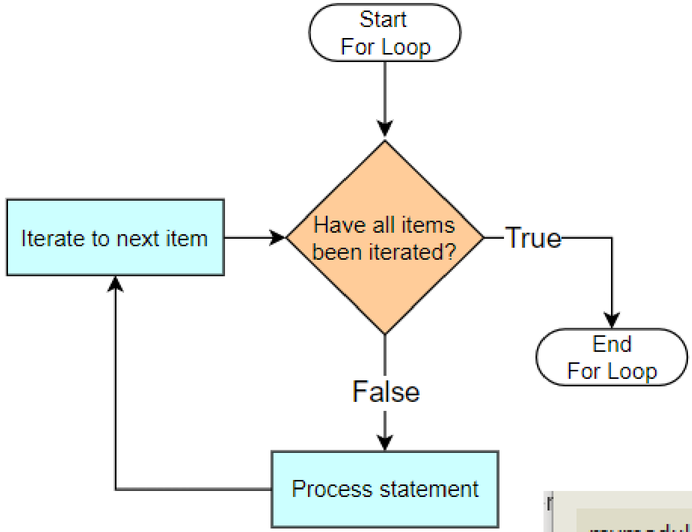


While Loop


Start
For Loop
Iterate loop 1 reinitialize loop 2
Have all items of Loop 1 been iterated?
True
Iterate loop 2
False
Process statement
False
Have all items of Loop 2 been iterated?
True
End
For Loop

Nested (for) Loop

For Loop



Start For Loop

Have all items been iterated?

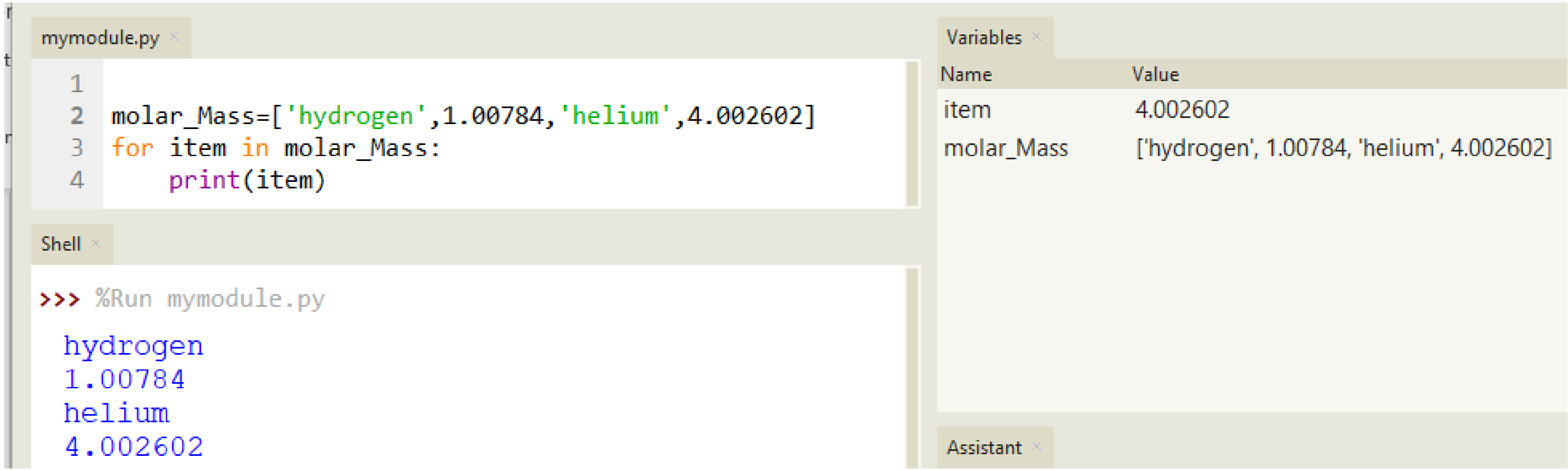
True

End For Loop

False

Process statement

Iterate to next item

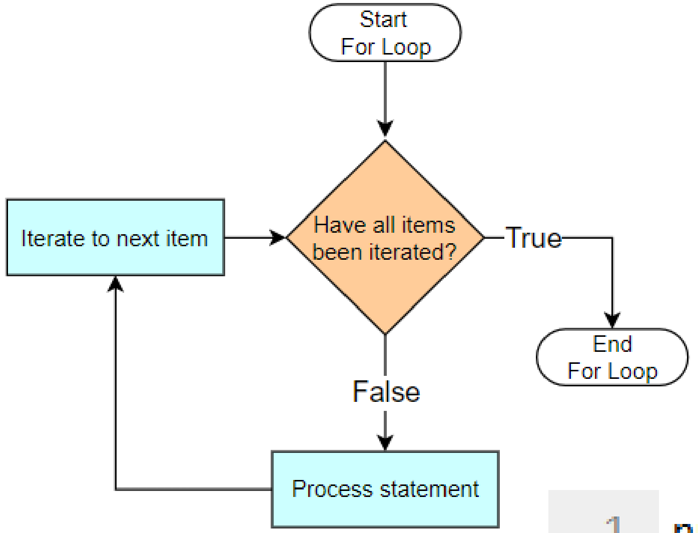


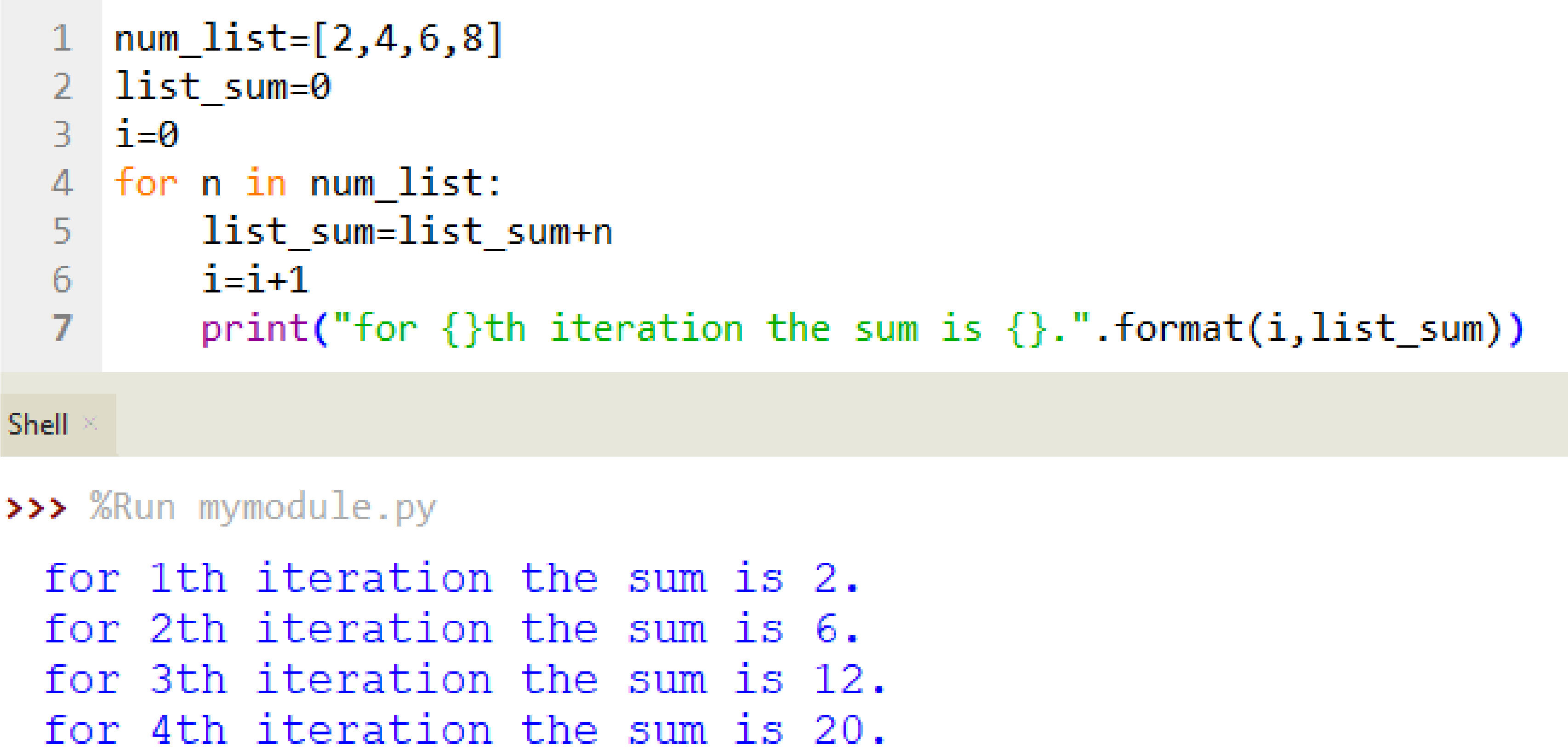
```python

molar_Mass=['hydrogen',1.00784,'helium',4.002602]
for item in molar_Mass:
    print(item)
```

```
>>> %Run mymodule.py
 hydrogen
 1.00784
 helium
 4.002602
```

Name	Value
item	4.002602
molar_Mass	['hydrogen', 1.00784, 'helium', 4.002602]

For Loop (for _ in)

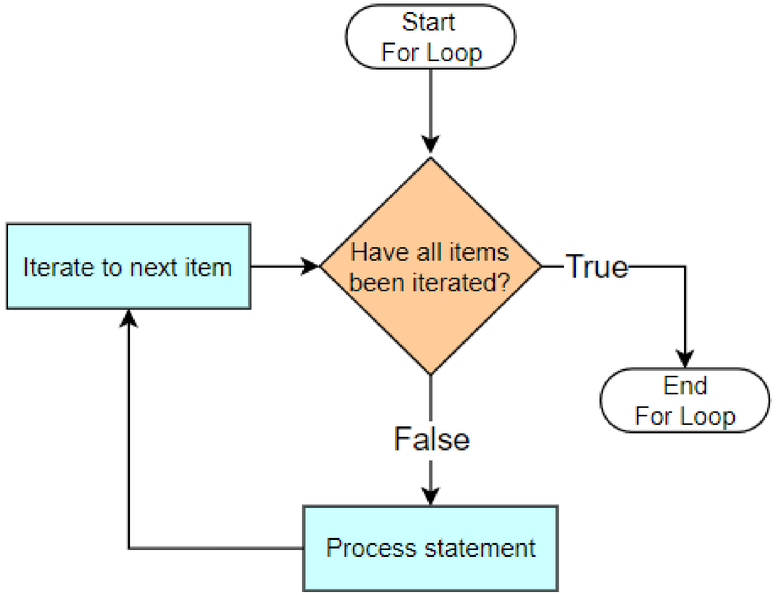


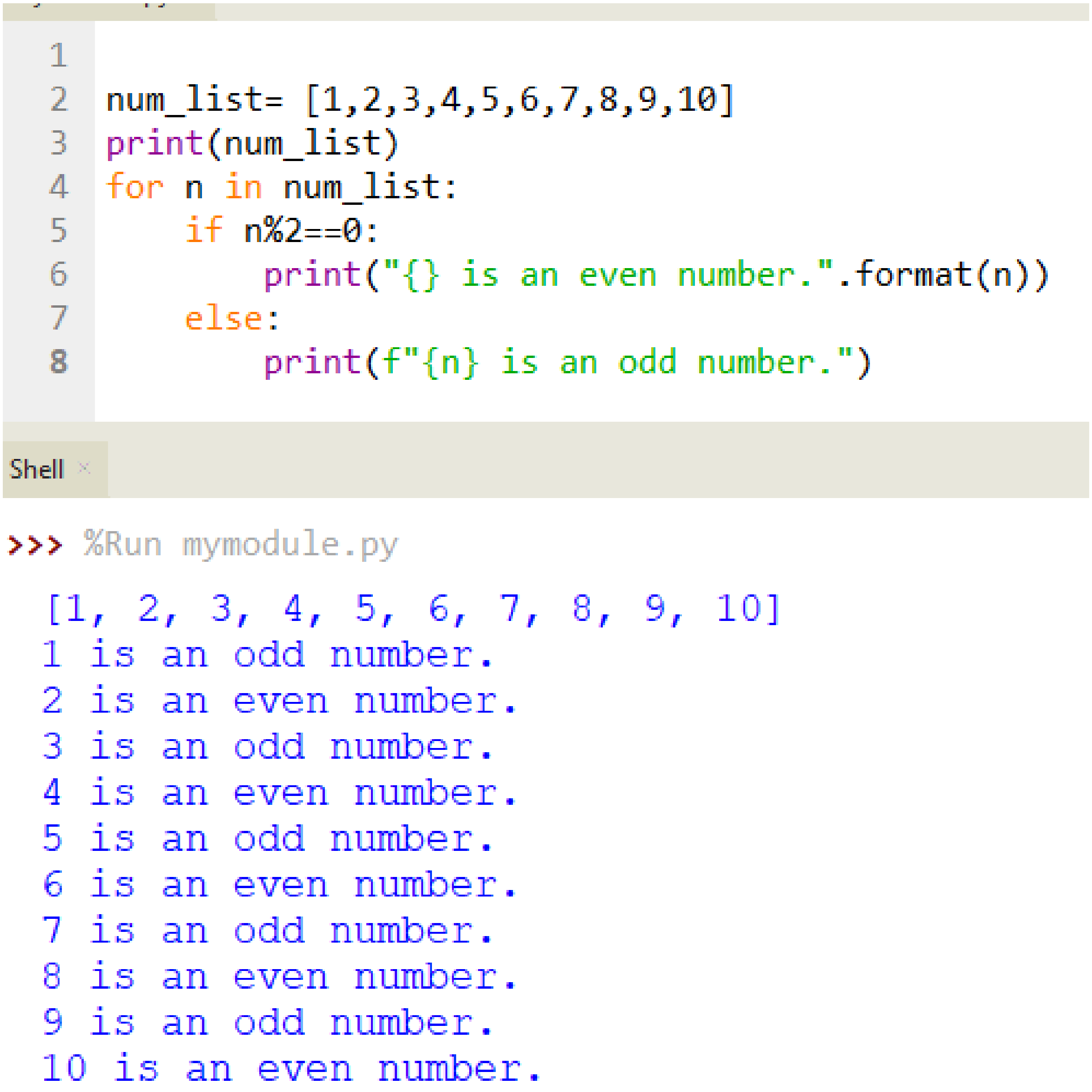


```
num_list=[2,4,6,8]
list_sum=0
i=0
for n in num_list:
    list_sum=list_sum+n
    i=i+1
    print("for {}th iteration the sum is {}.".format(i,list_sum))
```

```
>>> %Run mymodule.py

  for 1th iteration the sum is 2.
  for 2th iteration the sum is 6.
  for 3th iteration the sum is 12.
  for 4th iteration the sum is 20.
```

For Loop

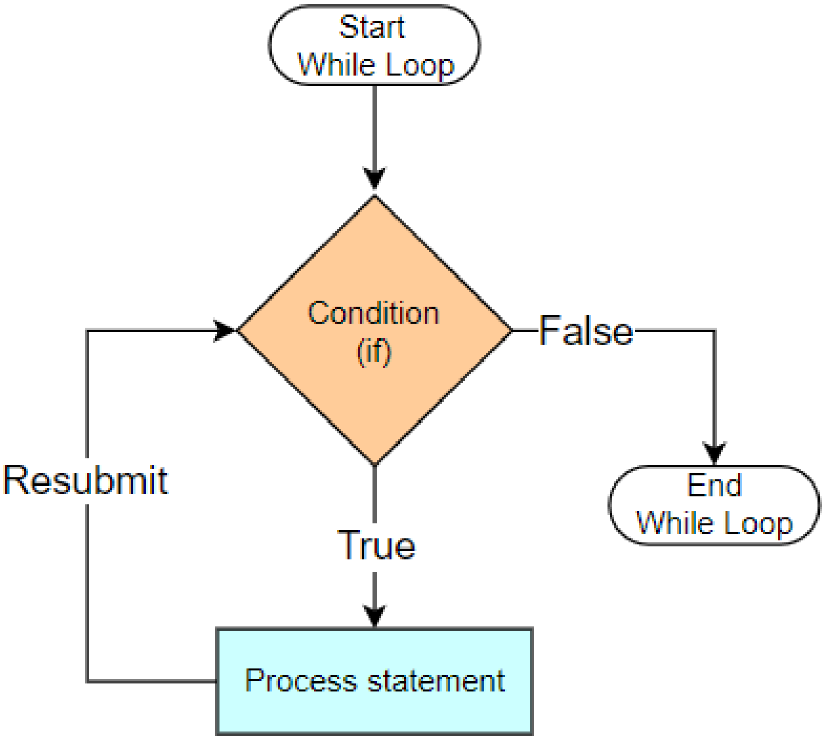


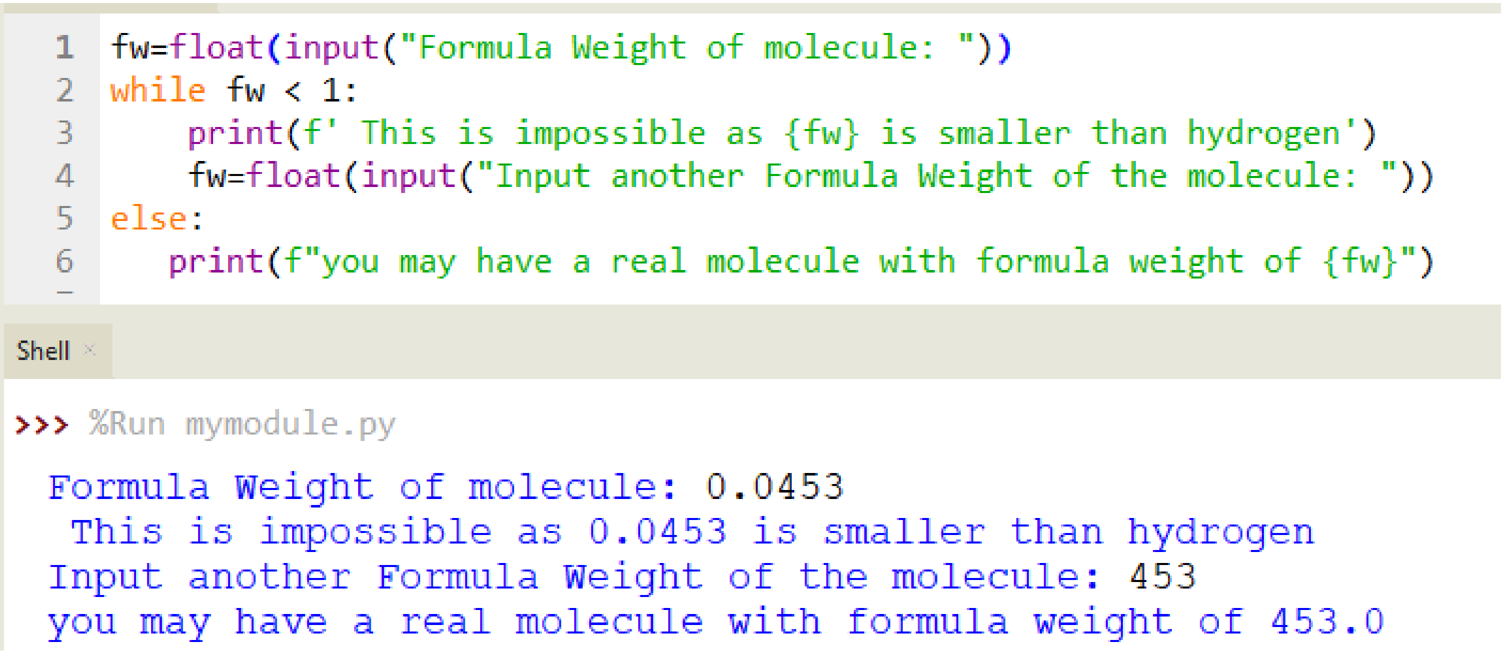


```python

num_list= [1,2,3,4,5,6,7,8,9,10]
print(num_list)
for n in num_list:
    if n%2==0:
        print("{} is an even number.".format(n))
    else:
        print(f"{n} is an odd number.")
```

```
>>> %Run mymodule.py

 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
 1 is an odd number.
 2 is an even number.
 3 is an odd number.
 4 is an even number.
 5 is an odd number.
 6 is an even number.
 7 is an odd number.
 8 is an even number.
 9 is an odd number.
 10 is an even number.
```

While Loop

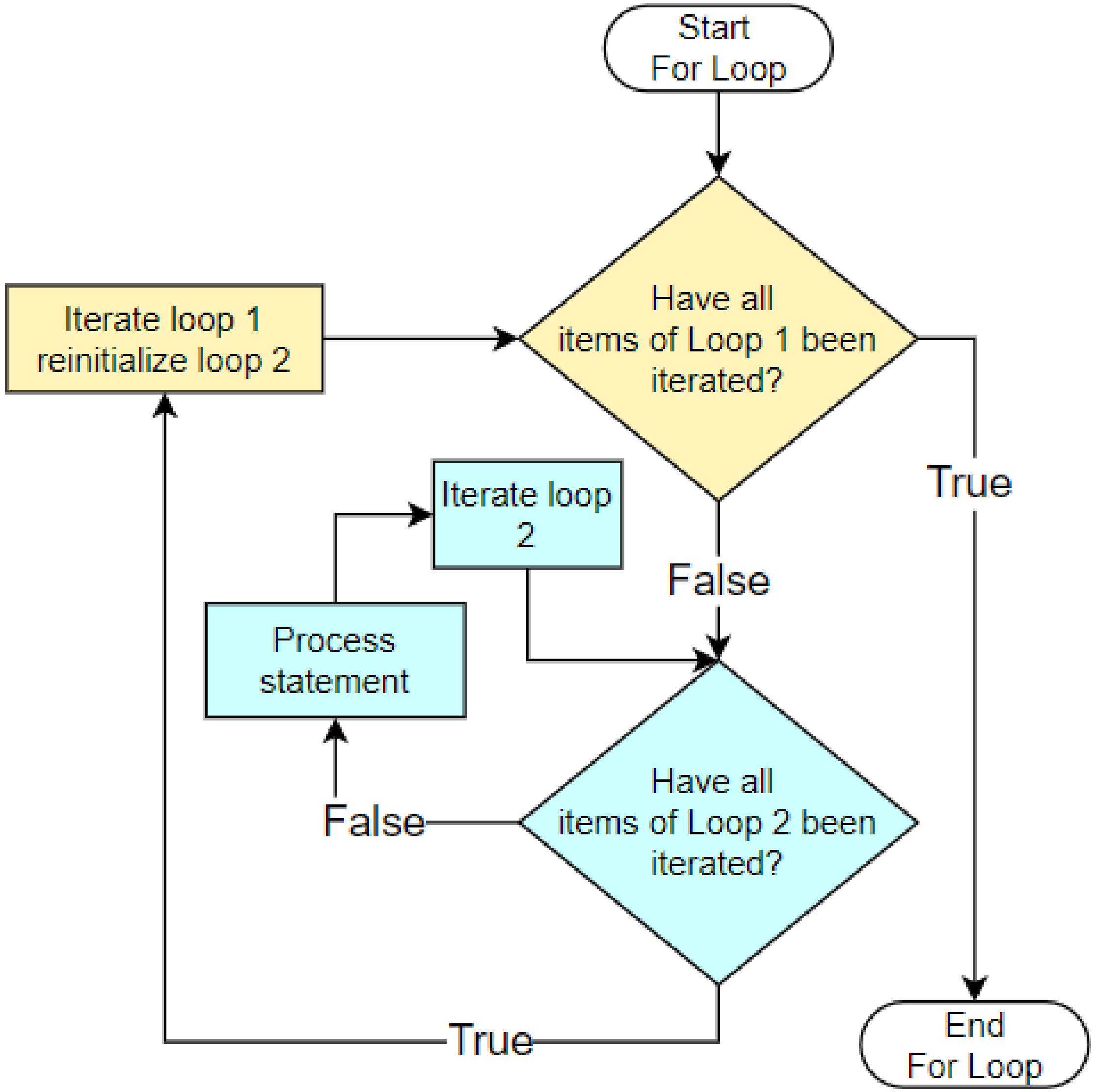


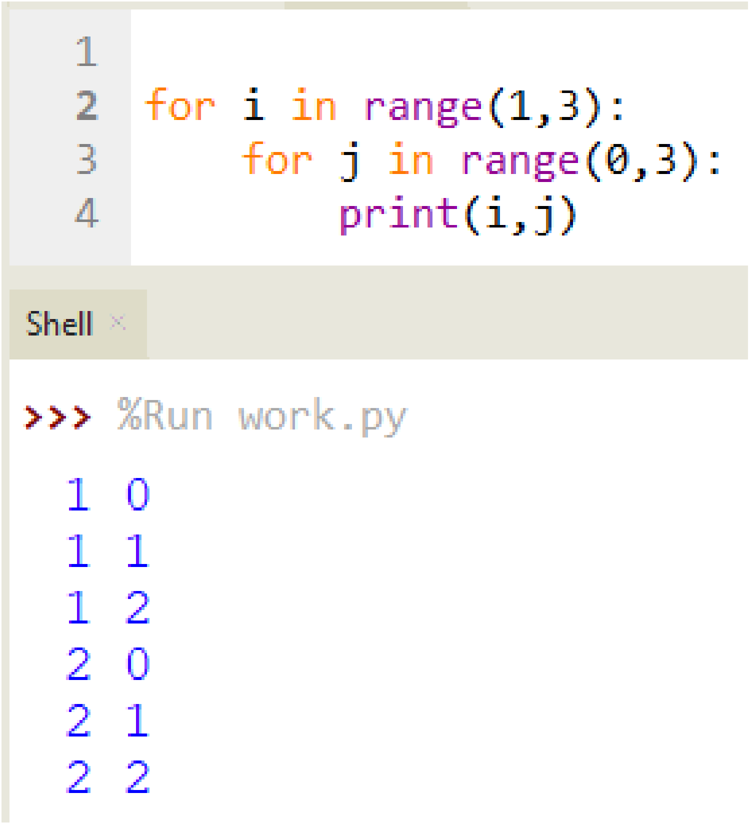


```
fw=float(input("Formula Weight of molecule: "))
while fw < 1:
    print(f' This is impossible as {fw} is smaller than hydrogen')
    fw=float(input("Input another Formula Weight of the molecule: "))
else:
   print(f"you may have a real molecule with formula weight of {fw}")
```

```
>>> %Run mymodule.py
 Formula Weight of molecule: 0.0453
  This is impossible as 0.0453 is smaller than hydrogen
 Input another Formula Weight of the molecule: 453
 you may have a real molecule with formula weight of 453.0
```

Nested Loop


Start
For Loop
Have all items of Loop 1 been iterated?
Iterate loop 1 reinitialize loop 2
True
False
Iterate loop 2
Process statement
Have all items of Loop 2 been iterated?
False
True
End
For Loop



```

for i in range(1,3):
    for j in range(0,3):
        print(i,j)
```

```
>>> %Run work.py

  1 0
  1 1
  1 2
  2 0
  2 1
  2 2
```