PHYS 214 Python Workshop

C.A.R.E. Tutoring

Please sign into the Queue

Workshop Objectives

1. Basics of Python and SymPy
2. Solving equations symbolically
3. Performing derivatives and integrals
4. Basics of NumPy
5. Applying these skills to PHYS 214 problems

Python Basics - Variables and Data Types

- Integers: x = 5
- Floats: y = 3.14
- Strings: course = “Thermal Physics”
- Booleans: is_warm = True

```
x = 5
y = 3.14
course = "Thermal Physics"
is_warm = True

print('Integer: x =', x)
print('Float: y =', x)
print('String:', course)
print('Boolean: is_warm =', is_warm)
```

```
Integer: x = 5
Float: y = 5
String: Thermal Physics
Boolean: is_warm = True
```

Python Basics - Math Operations

```
a = 5
b = 3

print('a + b =', a + b) #addition

print('a - b =', a - b) #subtraction

print('a * b =', a * b) #multiplication

print('a / b =', a / b) #division

print('a^b =', a**b) #exponentiation
```

```
a + b = 8

a - b = 2

a * b = 15

a / b = 1.6666666666666667

a^b = 125
```

Introduction to SymPy

- A Python Library for symbolic math

```
from sympy import *
```

- Can handle algebra, calculus, logarithms, and more

Defining Symbols and Expressions

- Create an expression that represents:
  - $2x^4 + 3y + 7$

```
x, y = symbols('x, y')
expr = 2*x**4 + 3*y + 7
expr
```

$$2x^4 + 3y + 7$$

- Create an expression that represents:
  - $x^3 + y^2 + 4x + 5y^8 + 2$

```
x, y = symbols('x, y')
expr2 = x**3 + y**2 + 4*x + 5*y**8 + 2
expr2
```

$$x^3 + 4x + 5y^8 + y^2 + 2$$

Specific Symbols and Functions in SymPy

Symbol/Function	SymPy Representation
π	pi
e^x	exp(x)
sin(x)	sin(x)
cos(x)	cos(x)
tan(x)	tan(x)
ln(x)	ln(x)
√x	sqrt(x)
∞	oo
x!	factorial(x)

Solving Equations

- Let's say we want to solve $x^2 - 4 = 0$:

```
eq1 = Eq(x**2 - 4, 0)
```
→ The comma (,) acts as the equal sign

```
soln = solve(eq1, x)
```
→ "Solve eq1 for x"

```
print(soln)
```
```
[-2, 2]
```

```
print('First Solution: x =', soln[0])
```
```
First Solution: x = -2
```

```
print('Second Solution: x =', soln[1])
```
```
Second Solution: x = 2
```

Solving Simultaneous Equations

- Let's say we have a system of equations that we want to solve:
  - 2x + 3y = 10 and -9x - 7y = 2

```
x, y = symbols('x, y')
eq1 = Eq(2*x + 3*y, 10)
eq2 = Eq(-9*x - 7*y, 2)

soln = solve((eq1, eq2), (x, y))
print(soln)
```

```
{x: -76/13, y: 94/13}
```

Solving Simultaneous Equations (Cont.)

```
x, y = symbols('x, y')
eq1 = Eq(2*x + 3*y, 10)
eq2 = Eq(-9*x - 7*y, 2)

soln = solve((eq1, eq2), (x, y))

x = soln[x]
y = soln[y]

print('x and y manipulation:', '2x =', 2*x, 'and', 'y/3 =', y/3)
```

```
x and y manipulation: 2x = -152/13 and y/3 = 94/39
```

Common Errors

- Undefined Symbols:
  - NameError

```
eq = Eq(2*x + 3*y, 10)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-a5175d72df30> in <cell line: 0>()
----> 1 eq = Eq(2*x + 3*y, 10)

NameError: name 'x' is not defined
```

- Fix:
  - Define symbols first

```
x, y = symbols('x y')
eq = Eq(2*x + 3*y, 10)
```

Common Errors (Cont.)

- No solution exists:
  - SymPy returns [ ]

```
eq1 = Eq(x + y, 3)
eq2 = Eq(2*x + 2*y, 7)

soln = solve((eq1, eq2), (x, y))
print("Solution:", soln)
```

```
Solution: []
```

- Many possible fixes:
  - Analyze the format and syntax of each equation
  - Think about whether it makes sense for there to be no solution [ ]

Calculus in SymPy

Symbol	SymPy Representation	Description
$\frac{d}{dx}$	diff(expr, x)	Derivative with respect to x
$\int dx$	integrate(expr, x)	Indefinite integral
$\int_a^b dx$	integrate(expr, (x, a, b))	Definite integral

Derivatives in SymPy (Cont.)

`diff(expression, variable, nth derivative)`

- Determine the 1st derivative of $x^3 + 2y^2 + 8xy$

```
x, y = symbols('x, y')
diff(x**3 + 2*y**2 + 8*x*y, x)
```

$$3x^2 + 8y$$

Derivatives in SymPy

`diff(expression, variable, nth derivative)`

- Determine the 2nd derivative of e^{-2x}

```
x, y = symbols('x, y')
diff(exp(-2*x), x, 2)
```

$$4e^{-2x}$$

Indefinite Integrals in SymPy - Example

```
integrate(expression, variable)
```

- Determine the integral of x^2:

```
x = symbols('x')
answer = integrate(x**2)
answer
```

$$\frac{x^3}{3}$$

Definite Integrals in SymPy - Example

```
integrate(expression, (variable, lwr bound, upr bound))
```

- Determine the integral of $2x + e^{-6x}$ from 0 to π:

```
x = symbols('x')
answer = integrate(2*x + exp(-6*x), (x, 0, pi))
answer
```

$$-\frac{1}{6e^{6\pi}} + \frac{1}{6} + \pi^2$$

```
x = symbols('x')
answer = integrate(2*x + exp(-6*x), (x, 0, pi))
answer.evalf()
```

$$10.0362710666706$$

Introduction to NumPy

- A Python Library for numerical computing

```
import numpy as np
```

- Can handle **complex numbers**, trigonometry, vectors, and more

Key Functions for Complex Numbers in NumPy

Function	Purpose	Example
`np.real(z)`	**Real part**	`np.real(3 + 4j) –> 3.0`
`np.imag(z)`	**Imaginary part**	`np.imag(3 + 4j) –> 4.0`
`np.conj(z)`	**Complex conjugate**	`np.conj(3 + 4j) -> 3 - 4j`
`np.abs(z)`	**Magnitude (modulus)**	`np.abs(3 + 4j) -> 5.0`
`np.angle(z)`	**Phase angle (in Radians)**	`np.angle(1 + 1j) -> 0.785`
`np.exp(z)`	**Polar → Rectangular form**	`np.exp(1 + 1j) -> 1.47 + 2.29j`

- **Avoid** writing 3 + *j**4 or 3 + *j*4 → Causes Python to treat *j* as a variable not the imaginary unit

Other Useful NumPy Functions

Function	Purpose	Example
`np.pi`	**Pi**	`np.pi –> 3.1415926535...`
`np.sin(z)`	**sin(z)**	`np.sin(np.pi) –> 0`
`np.cos(z)`	**cos(z)**	`np.cos(np.pi) -> -1`
`np.exp(z)`	e^z	`np.exp(0) -> 1`

Complex Numbers in NumPy - Example

- Let's determine the magnitude and polar form of the following complex number:
  z = 2 + 8j

Magnitude

```
import numpy as np

z = 2 + 8j
conjugate = np.conj(z)
magnitude_1 = (z*conjugate)**0.5
magnitude_2 = np.abs(z)

print('Complex conjugate =', conjugate)
print('Magnitude_1 =', magnitude_1)
print('Magnitude_2 =', magnitude_2)
```

```
Complex conjugate = (2-8j)
Magnitude_1 = (8.246211251235323+0j)
Magnitude_2 = 8.246211251235321
```

Polar Form

```
import numpy as np

z = 2 + 8j

theta = np.angle(z)
r = np.abs(z)
polar_form = r*np.exp(1j*theta)

print('r =', r)
print('theta =', theta, 'rads')
print('Polar Form: 8.246e^(1.326j)')
print('Verify output of polar form:', polar_form)
```

```
r = 8.246211251235321
theta = 1.3258176636680326 rads
Polar Form: 8.246e^(1.326j)
Verify output of polar form: (1.9999999999999993+8j)
```

Application to PHYS 214

- Calculating angles and lengths in slit diffraction:
  - sin(x) and asin(x)
- Using trigonometric identities (i.e. calculating the resulting intensity from wave superposition):
  - cos(x), acos(x), sin(x), asin(x)
- Later in the course:
  - Energy calculations - pi
  - Probability calculations - integrate(expr,(x,a,b)), numpy complex number functions
  - Optics/polarizers - cos(x)

Feedback Form