😊 Math and Random Modules

1. Python comes with a built-in math module and random module. In this lab, we will give a brief tour of their capabilities. Usually, you can simply look up the function call you are looking for in the online documentation.
2. We will start by importing math to our Notebook, then we will run help to know more about the math module in our Notebook.

```
[2]: import math

[4]: help(math)
```

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

        The result is between 0 and pi.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.
```

3. After that we started by **rounding off the values**. For a value of 4.35, using **math.floor** returns the largest whole number not greater than 4.35 (which is 4), **math.ceil** returns the smallest whole number not less than 4.35 (which is 5), and round rounds 4.35 to the nearest integer (which in this case is 4).

```
[6]: value = 4.35

[8]: math.floor(value)

[8]: 4

[10]: math.ceil(value)

[10]: 5

[12]: round(value)

[12]: 4
```

4. Below are some mathematical constants. When you run these commands in a Jupyter Notebook, you are accessing several constants from Python's math module:
    - **math.pi** is the constant π (approximately 3.14159), the ratio of a circle's circumference to its diameter.
    - Using "**from math import pi**" makes pi available directly in your code.
    - **math.e** is Euler's number (approximately 2.71828), the base of natural logarithms.
    - **math.tau** represents 2π (approximately 6.28318), which is the ratio of a circle's circumference to its radius.
    - **math.inf** is a representation of positive infinity.
    - **math.nan** stands for "Not a Number," used to denote undefined or unrepresentable numerical results.

```
[14]: math.pi

[14]: 3.141592653589793

[16]: from math import pi

[18]: pi

[18]: 3.141592653589793

[20]: math.e

[20]: 2.718281828459045

[22]: math.tau

[22]: 6.283185307179586

[24]: math.inf

[24]: inf

[26]: math.nan

[26]: nan
```

5. Now we are looking at some Logarithmic values. Here's what happens with each command:
    - **math.e** returns Euler's number (approximately 2.71828).
    - **math.log(math.e)** computes the natural logarithm of math.e, which equals 1 because the logarithm of e to base e is 1.
    - **math.log(0)** raises an error because the logarithm of 0 is mathematically undefined (it causes a math domain error).
    - **math.log(10)** computes the natural logarithm of 10 (approximately 2.302585092994046).

- **math.e ** 2.302585092994046** calculates e raised to the power of the natural logarithm of 10, which results in 10.

```
[28]: math.e

[28]: 2.718281828459045

[30]: # Log Base e
      math.log(math.e)

[30]: 1.0

[32]: # Will produce an error if value does not exist mathmatically
      math.log(0)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[32], line 2
      1 # Will produce an error if value does not exist mathmatically
----> 2 math.log(0)

ValueError: math domain error
```

```
[34]: math.log(10)

[34]: 2.302585092994046

[36]: math.e ** 2.302585092994046

[36]: 10.000000000000002
```

6. This confirms that logarithms and exponentiation are inverse operations.
   - **math.log(100, 10)** calculates the logarithm of 100 with base 10, which results in **2**.
   - **10**2 computes 10^2, which also results in **100**.

```
[38]: # math.log(x,base)
      math.log(100,10)

[38]: 2.0

[40]: 10**2

[40]: 100
```

7. The operations performed involve trigonometric calculations and angle conversions, where the sine of an angle is computed using radians, an angle is converted from radians to degrees for easier interpretation, and another function converts degrees to radians since many mathematical and programming functions rely on radian measurements for accurate calculations.

```
[42]: # Radians
      math.sin(10)

[42]: -0.5440211108893698

[44]: math.degrees(pi/2)

[44]: 90.0

[46]: math.radians(180)

[46]: 3.141592653589793
```

8. Random Module allows us to create random numbers. We can even set a seed to produce the same random set every time.
9. Setting a seed allows us to start from a seeded psuedorandom number generator, which means the same random numbers will show up in a series.
10. Note, you need the seed to be in the same cell if your using jupyter to guarantee the same results each time.
11. Getting a same set of random numbers can be important in situations where you will be trying different variations of functions and want to compare their performance on random values, but want to do it fairly (so you need the same set of random numbers each time).
12. This code demonstrates the use of Python's random module to generate random integers within a specified range. The **random.seed(101)** function ensures reproducibility by initializing the random number generator with a fixed value, meaning every time the code runs with the same seed, it will produce the same sequence of random numbers.

```
[48]: import random

[50]: random.randint(0,100)

[50]: 22

[52]: random.randint(0,100)

[52]: 30

[54]: # The value 101 is completely arbitrary, you can pass in any number you want
      random.seed(101)
      # You can run this cell as many times as you want, it will always return the same number
      random.randint(0,100)

[54]: 74

[56]: random.randint(0,100)

[56]: 24

[58]: # The value 101 is completely arbitrary, you can pass in any number you want
      random.seed(101)
      print(random.randint(0,100))
      print(random.randint(0,100))
      print(random.randint(0,100))
      print(random.randint(0,100))
      print(random.randint(0,100))

      74
      24
      69
      45
      59

[60]: random.randint(0,100)

[60]: 6
```

13. First created a list that ranges from 0 to 20 in numeric then we printed out our list. After that, we ran the random choice command to get a random number from our list.

```
[62]: mylist = list(range(0,20))

[64]: mylist

[64]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

[66]: random.choice(mylist)

[66]: 16

[68]: mylist

[68]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

14. Take a sample size, allowing picking elements more than once. Imagine a bag of numbered lottery balls, You reach in to grab a random lottery ball, then after marking down the number, **you place it back in the bag**, then continue picking another one.

```
[70]: random.choices(population=mylist,k=10)
```

```
[70]: [4, 4, 5, 13, 4, 19, 1, 3, 1, 15]
```

15. Once an item has been randomly picked, it can't be picked again. Imagine a bag of numbered lottery balls, You reach in to grab a random lottery ball, then after marking down the number, you **leave it out of the bag**, then continue picking another one.

```
[72]: random.sample(population=mylist,k=10)
```

```
[72]: [11, 6, 15, 10, 7, 16, 12, 18, 13, 3]
```

16. Now we are just shuffling our list using the random shuffle module.

```
[74]: # Don't assign this to anything!
      random.shuffle(mylist)
```

```
[76]: mylist
```

```
[76]: [12, 7, 19, 11, 0, 3, 17, 8, 15, 4, 5, 18, 16, 10, 1, 6, 9, 14, 13, 2]
```

17. Below we have used two types of distribution Uniform distribution and Normal/Gaussian Distribution. .

Uniform Distribution

```
[78]: # Continuous, random picks a value between a and b, each value has equal change of being picked.
      random.uniform(a=0,b=100)
```

```
[78]: 0.6518601416265479
```

Normal/Gaussian Distribution

```
[80]: random.gauss(mu=0,sigma=1)
```

```
[80]: -1.206313719234682
```