High School with Python

Use the math you already know, or learn it, and learn learning, to get stronger in Python.

Use the Python you already know, or learned recently, to teach yourself more math.

Climb the spiral staircase to greater mastery of today's tools and technologies.

- Age range: Open to anyone wanting to combine working out in Python, learning the language, while using high school level math topics as "grist for the mill".
- Format: Home school format over Zoom. BYOD (bring your own device). Hands on. Be willing to install new software to your local machine. Be willing to share your screen on Zoom. Be willing to open free accounts in the cloud, and post accounts optionally.
- Make your own Jupyter Notebooks and share them with your peers.
- Collaborate on coding projects.

Are you reading this as a PDF document? Then chances are, it was generated by an operating system level process looking something like this:

In [1]:
```
jupyter nbconvert --to webpdf class_proposal.ipynb
```

Out[1]:
```
[NbConvertApp] Converting notebook class_proposal.ipynb to webpdf
[NbConvertApp] Building PDF
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 278293 bytes to class_proposal.pdf
```

Getting Jupyter Notebooks to render as PDFs, including with $\LaTeX$ and Unicode emoji, is cutting edge tech at the time of this writing (Fall, 2021).

As a webpdf, this document has no page breaks, so good luck trying to print it out. Adding page breaks to webpdfs is a good example of an ongoing community project.

In [2]:
```
import sys
print(sys.version)
```

Out[2]:
```
3.7.9 (default, Aug 31 2020, 07:22:35)
[Clang 10.0.0 ]
```

Examples of high school math topics:

- Functions and Relations
- Trigonometry
- Cryptography
- Pascal's Triangle and Binomial Distribution
- Figurate and Polyhedral Number Sequences
- $\LaTeX$ for Math Expressions
- Publishing to the Web or other eFormats
- Complex Numbers (complex plane)
- Vectors and Matrices (spatial geometry)
- Group and Number Theory topics
- Mandelbrot Set (fractals)
- Calculus (differentiating and integrating)

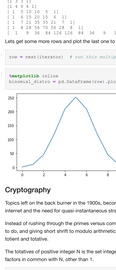
Trigonometry

Plotting sine waves in color requires very little code when you start with numpy and pandas. But then we're already inside a Jupyter Notebook, which you may think of as your new desktop workspace. In the context of JupyterLab, you will have easy access to a Python console, interactive editing, other file type viewing.

In [3]:
```
import numpy as np
import pandas as pd

%matplotlib inline

domain = np.linspace(-10, 10, 400)  # give me 400 points evenly spaced from -10 to 10
cos_wave = np.cos(domain)           # cosine; where x is every one of those 400 points
sin_wave = np.sin(domain)           # ditto sin(x)

# Build a table
tabular = pd.DataFrame({"domain": domain, "cosine": cos_wave, "sine": sin_wave})
```

In [4]:
```
# plot!
tabular.plot(x="domain", y=["cosine", "sine"], title="Trig Functions", grid=True)
```

Out[4]:



Pascal's Triangle

The Python *generator* below yields successive rows of Pascal's Triangle.

Generators are a special construct in Python. They're similar to functions but get nudged forward by next(), running until to a `yield` statement is encountered, restarting from wherever the program last yielded.

A generator does not forget its internal state, between nudges. Arriving at the same `yield` statement over and over, inside a while True loop, is not atypical, as in the example below.

In [5]:
```
def pascal():
    row = np.array([1])
    while True:
        yield row
        row = np.append(row, 0) + np.insert(row, 0, 0)

iterator = pascal()
for i in range(7):
    print(next(iterator))
```

Out[5]:
```
[1]
[1 1]
[1 2 1]
[1 3 3 1]
[1 4 6 4 1]
[1 5 10 10 5 1]
[1 6 15 20 15 6 1]
```

In [6]:
```
row = next(iterator)  # run this multiple times
```

In [7]:
```
%matplotlib inline
binomial_dists = pd.DataFrame(row).plot()
```

Out[7]:

Cryptography

Topics left on the back burner in the 1900s, become more front burner in the 2000s, with the rise of the internet and the need for quasi-instantaneous stranger-to-stranger secure transactions.

Instead of rushing through the primes versus composite distinction, as older high school curricula used to do, and giving short shrift to modulo arithmetic, we must now take a deeper dive, with notions of totient and totative.

The totatives of positive integer N is the set integers less than N, down to 1, including 1, that have no factors in common with N, other than 1.

In [8]:
```
from math import gcd

def totatives(N):
    return [n for n in range(1, N) if gcd(n, N) == 1]
```

In [9]:
```
totatives(9)
```

Out[9]:
```
[1, 2, 4, 5, 7, 8]
```

In [10]:
```
print(totatives(100))
```

Out[10]:
```
[1, 3, 7, 9, 11, 13, 17, 19, 21, 23, 27, 29, 31, 33, 37, 39, 41, 43, 47, 49, 51, 53, 5
7, 59, 61, 63, 67, 69, 71, 73, 77, 79, 81, 83, 87, 89, 91, 93, 97, 99]
```

Do not confuse totatives of N with prime factors of N, another topic we will be sure to take up.

The number of totatives a number N has, is called its totient. Euler sometimes used *φ* for his totient function.

In [11]:
```
def totient(N):
    return len(totatives(N))
```

In [12]:
```
totient(12)
```

Out[12]:
```
4
```

Group Theory

Totatives of N multiply as a Group modulo N. For example the 40 totatives of 100, multiplied modulo 100, have the properties of Closure, Associativity, Inverse and Neutral element (CAIN). This group is also Abelian. In this course, we will talk about what all that means, using a "math object" (type) called a Modulo Number. We'll also use Permutations.

In [13]:
```
totient(100)
```

Out[13]:
```
40
```

RSA

RSA public key *cryptography* is these days often only taken up in computer science after high school, but in this curriculum we bring in enough Group and Number Theory topics to make the RSA algorithm a door-opening application.

In [14]:
```
from primes import rsacrypto_test

Generating: p,q
p = 2147483647
Generating: q
c_hdbdq: 89x1iefteo8 n:11247x,e,d,p,q,b,q:private*

Python 3rd party community provides some useful cryptography packages. Lets used one.

The way RSA works is everybody publishes a huge composite number N, like an address, which has
two prime factors p, q that each person stores secretly.

There's no reasonable way to work backwards to get from the public composite number N, to its
component primes (p, q), given your current understanding of Number Theory.

Bob encrypts a message to Alice, by raising it in chunks to some power, with Alice's public key as the
modulus.

p = rsacrypto_test.p
P

1314348446046903694657420555993673235137966480339215497977267394764464305146980326917
8981279384841434063912666445569520267236733107385776080486680891

q = rsacrypto_test.q
Q

1148632362623001981767441999363872505432793006933211009423968764429531152974995605342
5937629524662305882546257524651154759326305171824247367031990595925

rsacrypto_test.e # raise to 17th power modulo p*q

17

Alice's secret "DNA-like" components then spring into action and decipher Bob's message, by raising it
to some still higher power, governed by the totient of p * q.

Alice can only decrypt messages intended for Alice (her public key address), and so on.

This system has worked well and is built in to every web browser as a way to handshake with public
keys, in order to shift to a faster private key, once a trusted connection has been established.

crypt = rsacrypto_test.rsaobj  # results of generating p, q, d

plain_text = b"no one will be able to read this without p, q"

cipher_text = crypt.encrypt(plain_text, b'X')[0]

print(crypt.decrypt(cipher_text))

b"no one will be able to read this without p, q"
```

Polynomials

If 1 is to a longer *ϕ*, as *ϕ* is to a still longer 1 = *ϕ*, then what is *ϕ*?

That question may be expressed as a solvable polynomial, yield a specific number, called the Golden Mean.

Notice $\LaTeX$ is being used to make our math expressions.

By doing the same thing to both sides, as we do with polynomials, we may simplify:

$$1/\phi = \phi/(\phi+1)$$
$$(\phi+1)/\phi = \phi$$
$$(\phi+1) = \phi^2$$
$$0 = \phi^2 - (\phi+1)$$
$$\phi^2 - \phi - 1 = 0$$

And next, we may solve this equation, using `sympy`.

In [15]:
```
from sympy import init_session
init_session()
```

Out[15]:
```
IPython console for SymPy 1.7.1 (Python 3.7.3-64-bit) (ground types: gmpy)

These commands were executed:
>>> from _ import *
>>> from sympy import *
>>> x, y, z, t = symbols('x y z t')
>>> k, m, n = symbols('k m n', integer=True)
>>> f, g, h = symbols('f g h', cls=Function)
>>> init_printing()

Documentation can be found at https://docs.sympy.org/1.7.1/
```

In [16]:
```
φ = symbols('φ')
```

In [17]:
```
from sympy.solvers import solve
```

In [18]:
```
solve(φ**2 - φ - 1, φ)
```

Out[18]:

$$\left[\frac{1}{2} - \frac{\sqrt{5}}{2}, \ \frac{1}{2} + \frac{\sqrt{5}}{2} \right]$$

In [19]:
```
sol = solve(φ**2 - φ - 1, φ)
sol[1].evalf(20)
```

Out[19]:
```
1.6180339887498948482045868343
```

This meaning for *ϕ* is distinct from Euler's use of it for "totient function" above. Math symbols get recycled in may pick up many meanings from different contexts.

The Golden Mean shows up everywhere in geometry.

Python

Python is a computer language, started in the Netherlands by Guido van Rossum in the 1990s, and still evolving.

Examples of Python topics:

- Using Python like a scientific graphing calculator
- Data Wrangling in Python (spreadsheets and databases)
- Jupyter Notebooks (like here)
- Coding with Functions (links to math)
- Object Oriented Programming (OOP)
- Dimensions of Python: keywords, built-ins, special names, library, 3rd party
- ASCII and Unicode

Surrounding Python is the broader landscape of the computerized office, with many components in the cloud such as:

Cloud Resources

- Google Colab (Notebooks)
- Replit (Computer Languages)
- Github (source under version control)
- PythonAnywhere (cloud hosting)
- Our Home Base Test (HBT)

The list continues. With the exception of PythonAnywhere, all of the above are free.

But what if we're not always connected to the internet, and what about working locally in a personal workspace, with files on the local computer (such as a laptop or desktop)?

That's where downloading and installation comes in. We focus on beefing up your local computer quite a bit, and do most depend solely on cloud resources.

Local Resources:

- JupyterLab (local version)
- Python / IDLE (the language itself)
- Git (source code under version control)
- Flask or Django or... (web framework)
- pip and conda (for downloading and installing)
- local copy of Home Base Test