for codepoint in range(int('1F600', 16) print(chr(codepoint), end="") EMOJI **⊕⊕⊕⊕⊕⊕₫₫₫⊕⊕⊕⊕**��⊖⊝₿ 13def hebrew(): for codepoint in range(int('05D0', 16), int('05DA', 16)) HEBREW print(chr(codepoint), end="") 16 GREEK & COPTIC for codepoint in range(int('03D0', 16), 19 ϐϑϒϒϔφͲϗϘϙϹϛϜϝϧϛϠϠϢϥϤϥϦͽϨϩϪϫϬϬͳϯϰϼϲϳϴͼ϶ϷϸϹϺϻϼϽϾ print(chr(codepoint), end="") 20 KORFAN 묀왽묂묃욀욁믦믧묈욅묊믦묌묍묎묏묐묑묒묓묔묕믶묗묘묙 for codepoint in range(int('BB00', 16), 묚묛묜묝묞묟묠묡묣묣묤묥묦묧묨묩묪묫묬묭묮묯묰묱묲묭 print(chr(codepoint), end="") 무묵묶묷문묹묺묻물묽묾뭃뭀뭂뭂뭃뭄뭅뭆뭇뭈믕뭋뭋뭌뭍 25 26def arabic(): print(" ".join([chr(codepoint) 묷풓뭐뭑뭒웏붠뭕뭕뭗풜뭙웛웗웘웙웛웛웜뭡뭢뤗윘뭥뭦웣 웝월월웨뭑둮퉦뭰뭕뭕뛤뮄뭙뭱뛢됈퉳뛢뫮뭰뭼뭰뭱뭿뙍뤵 for codepoint in range(int('0681', 28 뮂뮃뮄뮅뮆윃뮈뮉뮊뮋뮌뮍뮍윉뮏뮖뮑뮒뮓뮔뮕뮖윓뮘뮙뮚뮛 int('06AF', 뮜뮝윚윛뮠윝윞윟뮤뮥뮦뮧뮨뮩뮪뮫뮽뮭뮳 30 31def main(): 32 print("\nEMOJI") غ غ ع ع غ غ ع ع ڈ ډ ډ ڈ ڈ ڈ ڈ ڈ ڈ ز ر ر ر ر ر ر ز ژ ژ ښ ښ ښ ښ څ څ ځ ف ب ف ب ف ب ف ی ف ک ساگ ك ك ك ك emoji() Welcome! Welcome to the wonderful world of high school mathematics in conjunction with Python. We're going to be using the Python computer language to help understand some mathematics. My name is Kirby Urner and I'm the author of this text. So when you encounter me using the first person, you'll know of whom I speak. The Focus Imagine a futuristic high school that used a programming language to abet one's understanding of all matters mathematical, from polyhedrons to public key crypto. This would be a curriculum customized for such a school. This demonstration repo does not purport to completely cover all the territory it introduces. Use this text as a guide with many jumping off points. Also use it as exemplary of what your own demos might look like down the road, should you choose to adopt this technology. More specifically, this curriculum emphasizes Lambda over Delta Calculus, where the former is shorthand for discrete and/or digital mathematics. Delta Calculus is of course the current staple of any college prep diet. For a good example of strong curriculum writing focusing on Delta Calc with Python, check out A. Jorge Garcia's Shadowfax blog. Pedagogy My first foray into using Jupyter technology with high school students was in 2020, with the onset of the pandemic. I was working for Coding with Kids, driving to local schools and doing MIT Scratch, Codesters, Replit, and some other cloud-based, Chromebook-friendly stuff. However I was using Jupyter extensively with adults (see Andragogy), and I had also done an online camp or two, so Camp MightyMoose was born. Jupyter Notebooks are still a relatively new technology. There's a way to embed questions and run the answers through nbgrader. I imagine a school server hosting a growing collection of such Notebooks, some designed to be gradable, others available to be run and modified. I'm currently looking at Replit's Teams for Education as another core tool that might be used right alongside these Jupyter Notebooks. Andragogy Andragogy is "pedagogy for adults" meaning we don't teach people the same way regardless of age level and/or prior background. I've made a specialty of "Pythonic Andragogy" which is a pithy way of putting in a nutshell what I do in the "curriculum design" department. Don't be put off by the "high school" moniker / motif around here, as exclusionary. In my ideal civilization (in my "utopia"), adults get to revisit high school from time to time, meaning over a life span we need to get back to basics occassionally. In practice, many adults get to revisit high school vicariously, by helping kids with their homework. I'm not suggesting there's anything wrong with or obsolete about that model. On the other hand, older folks need to get new knowledge and cues from younger folk as well. Math Objects In case you're not seeing how an object oriented general purpose computer language might be relevant to learning math, lets briefly consider the idea of "math objects". When you want to understand how a radio works, it helps to build a radio. Build the machinery you're studying. Apply this same "maker" idea to something more metaphysical (or psychological) and you have "Making Math" a term we used around the O'Reilly School of Technology, then home to Make: Magazine. Consider the Rational Number type, consisting of two integer type numbers, the numerator and the denominator, thereby expressing a "ratio" (hence "rational"). What must we do when adding or subtracting two fractions, or multiplying and dividing? How about taking the reciprocal or inverse? Raising to a power? In Python, we have operator overloading, meaning you, the coder, are free to spell out the meaning of +, -, /, *, ** vis-a-vis your specific type of object. You get to build the Rational Number type, the Permutation type, the Mod type which does its operations relative to some modulus (see nums.py). The H Shape Think of Python and Mathematics as the two sides of a ladder, with rungs connecting them. In climbing this ladder, you use your math to learn Python, and your Python to learn math. The Topics As previously mentioned, a way of circumscribing a set of topics is to set a boundary criterion, namely that of "relevant to high schoolers". One way of implementing that filter is by using past, existing models, as a guide, and certainly some of that is going on here. I've been a high school mathematics teacher in a prior decade (1980s). However, having grown up in a household subscribed to *The Futurist*, with a dad who was all about planning as a profession, I'm mostly about encountering new horizons. My treatment of the Mandelbrot Set in the Argand Plane, of Cryptography, and finally of spatial geometry, with so-called Quadrays atop XYZ, will impress many of you as either intelligent foreshadowing, long overdue innovations, or quirky expressions of an overly idiosyncratic point of view. All quirkiness aside, I'm assuming we all agree that high school level mathematics includes complex numbers, some trigonometry, and at least some single variable calculus. I work to cover these bases, among others, except by "cover" I mean "suggestively touch upon" because ultimately my aim is to inspire additional and deeper treatments of the included topics. The Computer Language Python was Made in The Netherlands (North and South Holland are but provinces) and enhanced by a global team. Guido van Rossum invented the language and guided its progress as BDFL. The language continues to evolve. BDFL means Benevolent Dictator for Life and was Guido's snarky title, in the same comic tradition as "world domination" being the unofficial goal of Linux + GNU. Not that this goal went unreached, in the form of self mastery. Guido named his language after Monty Python, the BBC comedy troupe that also became popular in America, despite the censorship and dumbing down for prime time US TV. However, there's no ignoring that a python is likewise a snake, important in Greek mythology. Python's evolution is governed by a sophisticated process of submitting Python Enhancement Proposals (PEPs), which are enumerated and considered, much like bills before a congress. Debate is mostly among those most invested, as developers of the core language. Python's story begins in the 1990s, with Guido a mathematics major fresh from Amsterdam, and joining Stichting Mathematisch Centrum, a Dutch think tank. The institute was already using a REPL (pronounced "rehpull") inhouse, called ABC, and ABC along with C, ML, some other languages, fed Guido's quest for a new computer language for everybody, yet not especially for beginners (not another BASIC). REPL means Read Evaluate Print Loop, and describes the workflow of any interactive shell, where the user types, the computer reads and evaluates, prints a result, and waits for a next user command. Python, like APL, like xBase (dBase, FoxPro) with its "dot prompt", is interactive (conversational) right out of the box (upon bootup). You can use it like a calculator. What you are looking at now is a Jupyter Notebook file transformed, on the fly, by a rendering engine, into HTML, which in turn instructs your browser in how to render the page in its final form. import sys sys.version '3.7.9 (default, Aug 31 2020, 07:22:35) \n[Clang 10.0.0]' Do you want to run multiple versions of Python on the same computer? There are ways... Python comes with a "virtual environment system" for example. The Standard Library includes venv. The Anaconda distribution of Python includes its own virtual environment solution. Let's exercise a built-in, something Python comes with, built in, minus any need to use the keyword import. copyright Out[2]: Copyright (c) 2001-2020 Python Software Foundation. All Rights Reserved. Copyright (c) 2000 BeOpen.com. All Rights Reserved. Copyright (c) 1995-2001 Corporation for National Research Initiatives. All Rights Reserved. Copyright (c) 1991-1995 Stichting Mathematisch Centrum, Amsterdam. Course Framework The main tools we'll be using: Python: general purpose computer language • an IDE (at least one): development environment · Github / Git: version control and sharing JupyterLab: includes Jupyter Notebooks + IDE Replit: Python in the cloud • Google Colab / Saturn Cloud: Notebooks in the cloud Typically a student will have a fairly beefy localhost laptop able to run Python and Jupyter locally. I've been recommending the Anaconda distribution because it already contains, along with Python, 3rd party tools such as we use in this course, including JupyterLab and the Spyder IDE. 4dsolutions/elite_school

/... (other repos)

CLONE/ZIP

Pulh STUDENT LAPTOP (or Decktop...) **Python ₽** python™ Python was not designed for beginners, but it turned out simple enough to introduce the Object Oriented Paradigm (OOP). Python has functional aspects too, meaning functions outside of classes make perfect sense, unlike in Java. IDE An Interactive Development Environment is about making your life as a coder a lot easier. The interactive shell or REPL, text editor, debugger, linter, spellchecker, all come together in a pleasant dashboard. Don't feel like you always have to use the same IDE. Try a few of them. **M**CODE Visual Studio Code Some IDEs are language specific, such as Spyder and IDLE, whereas others give you optional plug-ins or add-ons (extensions) for working with multiple languages (vscode is of this type, as are Emacs and Eclipse). One of the minimal IDEs is Vim, based on Vi. Vi grew up in the UNIX environment which also features what are called Regular Expressions, used for pattern matching, as when searching text. Many UNIX commands feature them, such as grep. Python has a decently endowed regular expressions library (named re) but does not make "regexes" (for short) as central to the language as Perl did. Perl was already well established when Python was born. O'Reilly's Open Source Conference (OSCON) was still the Perl Conference. These happen in Portland (usually). Github / Git Version Control with O'REILLY' Here's where Youtube or another source of videos might come in handy, to explain the world of Version Control, both in broad brush strokes and up close. Git arose from the GNU Linux Project. Linus Torvalds needed a way to keep the foundations under Linux, its version control system, collaboratively used from around the world, free and clear of future "right to use" obstructions. In other words, the version control system used for Linux was too important a dependency to leave to others. Picture many people working on a project and wanting to take it in various directions, provisionally. You have a main trunk with experimental branches, with the opportunity to merge successful threads back into the main action. That's one pattern for using Git. The main point of version control is you're able to roll back through time and use snapshots of how the project looked in the past. Drafts get saved. Changes stay undoable, at least in theory. Heads Up As of this writing, Github is changing the workflow such that a specific session token, rather than a user password, is needed to post changes. JupyterLab 5/biob/menterSTEM Mathem 日 ♥ 9, Search 本 音 ☆ 白 〇 日 4 巻 〇 〇 田 jupyter JUPYTER FAQ III O Crystal Ball Sequence The face-centered cubic (PCC) lattice is not always presented in this simplest form, ditto the cubic close packing (CCP), which amounts to the same thing. A nuclear ball is surrounded by a layer of serior, a flooring it, and adjacent neighbors. The shape so formed in not a cubic but a cubic-balledrow, with eight fraingular faces and six square. As the cuboctahedral packing continues to expand outward, layer by layer, the cumulative number of balls or points fo cubocta(), a generator, yields the number of balls in each successive layer of the cuboctahedron, according to a simple formula derived by R. Buchminster Fuller, a prollific inventor and philosopher [1], cummulative() delegates to cubocta() white accumulating the number in each layer to provide a numing total. In [1]: from itertools import accumulate, islice Classic Generator: Cuboctahedral / Icosahedral #s https://oeis.org/Add5901 The Jupyter Project arose from the I-Python Notebook Project, which in turn arose from the IPython Project. JupyterLab is a browser-based IDE for Jupyter Notebooks, one of which you're looking at here (unless you're reading a PDF derived from it). The main text for our courses is likely in this format (ipynb), in our futuristic high school. Teachers and students alike author courseware and related projects using JupyterLab. Jupyter Notebooks implement what Donald Knuth, a father of computer science, dubbed "literate programming". Instead of just source code, with comments and docstrings sprinkled in, you have the ability to mix "code cells" with "markdown cells". The former include runnable source code, which produce output, whereas the latter communicate directly to humans via words, images, and links. What you are reading right now is a markdown cell. The first code cell on this page was nearer the top, where we queried Python for a version number. **Major Topics:** Object Oriented Programming or OOP Coding Through Storytelling DB API (talking to databases) numpy and pandas Algorithms in Cryptography sympy, a computer algebra system (CAS) Tractors in a Field (Python generators) Random Walks in the Matrix (Quadray Coordinates) Sub Topics: A Data Structure: the Binary Tree named tuples mutability versus immutability callability sorting sequences "randomness" slicing and dicing sequences Unicode date and time related topics ACSL exercises Study Sheets · Lambda Notation with map, reduce, filter Sigma Notation Pascal's Triangle and the Python generator · Converging to Phi by Continued Fraction Generating the Digits of Pi Logarithms Delta Calculus Trigonometry **Complex Numbers** Immutability versus Mutability The tuple of tuples below is read only. zoo = (("Ape", 4), ("Zebra", 2), ("Penguin", 1)) The tuple of lists below looks like it might be immutable but is it? In [4]: L = ([], [])# is L immutable? ([], []) Out[5]: L[1].append("dog") L # no! Out[6]: ([], ['dog']) Callability This topic is worth a whole page to itself. Sorting Sequences How might you sort zoo tuples by either animal or age? Here's a way I would not expect a beginner to find right away: from operator import itemgetter sorted(zoo, key=itemgetter(1)) [('Penguin', 1), ('Zebra', 2), ('Ape', 4)] sorted(zoo, key=itemgetter(0)) Out[9]: [('Ape', 4), ('Penguin', 1), ('Zebra', 2)] Randomness from random import choice, shuffle zoo Out[11]: (('Ape', 4), ('Zebra', 2), ('Penguin', 1)) shuffle mutates its argument, and since zoo is immutable, it's not a legal argument to shuffle. However, we may replace the outer tuple with a mutable list and then apply randomness. shake up = list(zoo)shuffle(shake_up) shake_up Out[12]: [('Zebra', 2), ('Ape', 4), ('Penguin', 1)] shuffle(shake_up) shake_up Out[13]: [('Ape', 4), ('Zebra', 2), ('Penguin', 1)] In [14]: choice(shake_up) # pick an element at random ('Ape', 4) Out[14]: def get_token(n=25): chars = "abcdefghijklmnopqrstuvwxyz1234567890" return "".join([choice(chars) for _ in range(n)]) get_token() '7eovprsbdrxod9lfmrls876c2' Slicing and Dicing Sequences Sequences (like lists, tuples, ranges) have left-to-right (right-to-left) order, are sortable and reversible. Mappings (dicts, sets), in contrast, are more general, in that the same content, in arbitrary order, connotes equality. {"a":"b", "c":"d"} == {"c":"d", "a":"b"} Out[16]: True ["a", "b", "c", "d"] == ["c", "b", "a", "d"] Out[17]: False set(["a", "b", "c", "d"]) == set(["c", "b", "a", "d"]) Out[18]: True Recent Pythons implement PEP 448, expanding the powers of the single- and double-star when used as prefixing unary packing / unpacking operators. (*zoo, *zoo) (('Ape', 4), ('Zebra', 2), ('Penguin', 1), ('Ape', 4), ('Zebra', 2), ('Penguin', 1)) a slice = slice(0, None, 2) (*zoo, *zoo)[a_slice] Out[20]: (('Ape', 4), ('Penguin', 1), ('Zebra', 2)) to_ship_tupl = (*zoo, *zoo)[a_slice] to_ship_dict = dict(to_ship_tupl) to_ship_dict Out[21]: {'Ape': 4, 'Penguin': 1, 'Zebra': When a dict cries out to be converted into named arguments, to any callable, the double-star unary operator, prefixed, comes to our aid. Likewise, as we learn when studying callability, the double-star operator, in front of a parameter name, will match any remaining named arguments to a dict, by that name. dict(**{"Llama":2}, **to ship dict) Out[22]: {'Llama': 2, 'Ape': 4, 'Penguin': 1, 'Zebra': 2} Unicode As we discussed in our first meetup, Unicode is the successor to ASCII both in the sense of superceding, and in the sense of swallowing and including. Python libraries will let programmers continue working in pure ASCII, or in any of several codecs whereby plain text is decoded and encoded. UTF-8 is the most common flavor of Unicode and is Python's default. The format commits varying numbers of bytes to the Unicode codepoints, with information up front (front byte) regarding how many bytes follow (picture a tiny train of one to four cars, car = byte). import fooding In [24]: fooding.foods Out[24]: ['\s\', '\omega', '\omega', '\gamma'] fooding.fruits Out[25]: ['\documents, '\documents, '\docume Fooding? Explorations with Unicode: The script below is designed to run at the command line, and therefore has to read arguments from the command line. unicode_fun.py Proposed project: Add another language section to the unicode_fun.py menu and to the output. Additional Resources: **Proposed Exercises** Youtube Gallery (curated) Curriculum Using Python + Jupyter Notebooks GeoPython by the Department of Geosciences and Geography at the University of Helsinki Python Data Science Handbook by Jake VanderPlas School of Tomorrow by Kirby Urner For Further Reading History of Python (my Wiki article) Generating the FCC computer graphics stuff Intro to Python Flickr Album used in courses Answering Questions on Quora (doing my civic duty) PyCamp: Another Repo for Teens by me Python Release Schedule (3.9, 3.10, 3.11...) In the Beginning Was the Command Line by Neal Stephenson, a classic Revolution OS movie documentary A Monty Python Story in the New Yorker (1976) Course Alumni (optional, self identified)