Elves()

a flexible python multidimensional-only array packer & unpacker

All array operations needed - in English

result = Elves.speak("English").(
variable
"add array 'd' to the end of array" ticks
"add string 'd' to the end of array —
"add int 'd' to the end of array —
"add float 'd' to the end of array —
)

I can write all my code like that
in a sequence - in repeating English code structure

Exactly what the crypto test will do in English & they will make it run

bag='napsack'

English sentence based coding

array = Elves.help_in_English("phrase")

array= Elves.~~hardlabel~~help(

"make an array with numbers 1 to 10"

)

A string tells the function what to do

for i in ~~range~~ range(20): #Get 20 ticks of crypto market data for a list of 20 of them: -BTC -ETH -ADA

for i in range(20)

c=~~list of~~ Coinbase('tick','ETH')

d=Elves.repack(c)("tp") # time & price only

~~tick~~ Elves.add(d) ticks) #append to ticks

array

append d to ticks[n]

(list expressing in English like this ONLY)

ENGLISH THEN

all array ~~operations~~ for crypto test

Elves.help_in_English("

"add array 'd' to the end of array ticks"

4) add string 'd' to the end of

add number 'd' to the end of array

add float 'd' to the end of array

for crypto test

All operations Needed in Elves-English

my new code language - all in English sentences

Elves.do_list.speak("English").(

crypto td("read the current tick data for these cryptos "ETH-US" for these cryptos: ETH, BTC, ADA")

tds "get tick data for all in 'cryptoList' array

tds_by_crypto_name_ID saves it like ['ETH'] array(time, price) 20-30 in there
['BTC'] array(time, price)
['ADA'] = array(time, price)
by index of crypto name ID

tick_data_series" ordered by (by crypto_name_ID)

ordered by 'name'
['name']
['ETH-US']

" Loop & get tick data series (20) ticks
In need of
with a (2) second pause
wait between them
for a list of crypto_name_IDs

E.speak("

ticks = " loop & get tick data. from coinbase_api.
Return
Give me a series of (20) ticks
with a (2) second wait in between ticks,

ticks2 = " Elves.speak("
Pull out time & price & return array with (time, price)

)

Elves . Speak - English sentence strings that do actions
"English"

in LIST, All cryptos

<LOOP> tick_from_crypto('ETH-US')
('BTC-US')
(~~'ETC-US'~~)
...

useful
u[] = array()

has 20 in list → stored as strings like ↑

ID_List = crypto_IDs

for ~~each every~~ ID in ~~crypto_IDs~~ ID-LIST

data = array
d = tick_from_coinbase_pro(ID) crypto Acronym

useful
~~u. Elves speak(#~~

u.append. ~~(...)~~ keep("time, price", d).save by ~~Key~~ Index (ID)
from Array

↑ append, but keep only 'time' & 'price' multidimens. parts
to u (useful)

(At the end u[] has all the useful)

d = tick_data_from_coinbase(ID)
u.append. Keep("time, price", d).saveBy Key(ID)

↓

story

Metaphor coding

sample 1 sentence story for Elves()

"Elves() pack() their leaf bags satchels for the long journey ahead"

Based on mythology for Elves

Here is the story

↑

the code will look like this story.

STYLE OF CODE FORMAT & COMMANDS & FUNCTION NAMES

so we can use variations of the word 'pack' like re-pack or un-pack

you could use unpack '&' then the official name for an elves' backpack - if there is one

we could stick to Elves Lore

Elves() = Array Packer (mostly)

~~Cave Sketches 2.5~~

array = Elves.pack(fruits)
↑ fruit names (string)

array2 = Elves.rePack(array, parameters)
↑ how to repack it

it sets everything up as a multidimensional array

Elves() = multidimensonal array

(1) packer() & (2) repacker()

↗ creates the array

↑ creates an new adjusted array based on 1 (or more) -later-

= I do the high level organizing =

they help me write it

Elves()

E.AddTo('name', 'value')

so you can do this in PHP

$addTo = array(0 = 'hi', 1 = 'hello')

addTo[0] = 'hi'

addTo[1] = 'hello'

simple way to assign

Python needs a simple way to add to this type of array

'EHI' = array[-]
'GTC' = array[-]
'ADA' = array[-]

array = (columns: trade-id, price, size, time, bid, ask, volume)

array2 = Elves(array1, "use columns['time', 'price']")

this make a new array - with only those columns of data

array = Elves.NewArrayBasedOnColumnNamesChosen('-', '-')

a few of these

names of columns

Array functions

Elves()

how will they help?

with managing

for crypto tick data

- crypto abbreviated name
- time - that sec or millisec
- price

build a new array - structured visually good

w

[0] sec [1] sec [2] sec ← array.

sec[N] = { name price time }

[time-second] = array (

CryptoElves[] = array (

'ETH'

'BTC'

'ADA'

)

)

then how to access it

Cool

strings → Elves.Examine(string1, string2, "phrase") time-string float price string

Elves.examine() #compares strings by english phrases

check = Elves.examine(price1, price2, comparison-phrase: "did it go up")

float float

if check = True:
    print("price went up")

Indent →

percent = Elves.examine(price1, price2, comparison-phrase "what percent did it go up")

print(str(percent) + "%")

the compary command on → comparison phrase

time_passed = Elves.examine(time1, time2, "how much time has passed in seconds")

test_bank account = 500.00

testbuy $10 of each crypto

get the ratio: their coin to $1 → # of their coins

NUM_of_coins = 1.32

coin_ratio_to_1_USD = float (10 decimals places on float)

get value difference → Time X D

for $10

functions that easily convert cryptos - currencies - coins - amounts - based on current coin price

some cryptos the coins you get are 1000's & others are a small fraction

```
import Elves
                                    Array of strings
                                          ↓
ID_list = crypto_IDs = ('ETH-US', 'BTC-US',)
ticks = array()
for n in range(20):
    for ID in ID_list:
        ticks[ID][n] = coinbase.tick(ID).
                        .keep_only('time','price')


print(ticks)
```

'ETH-US': 1: 'time': 11:59:01, 'price': 1.30
2: 'time': 11:59:02, 'price': 1.33
3: 'time': 11:59:03, 'price': 1.36
4: 'time': 11:59:04, 'price': 1.43
5: 'time': 11:59:05, 'price': 1.52
...
20: 'time': 11:59:20, 'price': 1.56

'BTC-US': 1: 'time': 11:59:01, 'price': 34.30
2: 'time': 11:59:02, 'price': 34.37
3: 'time': 11:59:03, 'price': 35.20
4: 'time': 11:59:04, 'price': 34.21
5:
...
20:

etc...

'...'

= etc...

describe what crypto() will do
~~& how elves() will help with array functions~~

ticks_array = E.speak("getting crypto list - 20 ticks each")

"which went up?" — the % it went up

produces an array:
array[went up] = (ETH-US, 2.32%), (BTC-US, 1.1%), ...

tick 0 + went up
[0] → [1]

array_up = elves.crypto('went up', price, 0, 1, ticks_array) + went up

tick[0] → tick[1]

array_down = elves.crypto('went down', price, 0, 1, ticks_array)

1st frame
1st tick of data → to 2nd tick

Returns output: structured like:

array-up	(ETH-US, +2.32%), (BTC-US, +1.1%),
array-down	(SHIB-US, -7.31%), (NU-US, -4.02%),

Testing Array Bank / Bank → Crypto → Array

calculate()

elves.crypto("how many coins is ETH-US if I spend $10")

elves.crypto.calculate()

how many coins for $10 US of ETH-US

Bank_Array()

Bank = $500

based on the % rise or fall of the tick data for a crypto

how much of the $10 is left? on the 2nd tick

bank(ticks(all) w/totals per tick) — per crypto

We buy at the 1st tick price $10 US of that crypto's coin

how_much_coin('ETH-US') do_I_have

~~total~~	~~total~~	~~501.25~~
$10	10.01	10.02
$10	9.99	10.01

what_is_the_US_Dollar_Value of my current test $10 buy of 'ETH-US'

every tick total - as it rises & falls

bank = array (ETH: [0] coins, value; BTC: coins, value)

1st [0]

ON the 2nd tick, 3rd, 4th

[1] [2] [3]

$10 buy in

$ value = price

$ Value? $Val ? $Val ?

how much is that $10 go up or down over each tick

20 ticks total

tick0 = value

tick1 = value +-

tick2 = value +-

tick3 = Value +-

tick data for 1 seand (need 20)

~~for N in range(20):~~

useful
~~U=usefularray~~ U[ ] = array

ID_list = crypto_IDs (strings ETH-US, BTC-US, ADA-US)

for ID in ID_list:

d = data_array = coinbase.tick(ID)

~~d.keep_only('time','price',d)~~

[ID][N] = d.keep_only('time','price')

~~ticks[ID]~~ ticks[ID][n] = d['time'] d['price']

~~print(d)~~

repeat this 20 X & save in ticks

~~print(d)~~

'ETH-US': time = 11:59:01 price = 1.30

'BTC-US': time = 11:59:01, price = 25.30

'ADA-US': time = 11:59:01, price = 3.30

↓

end: time, price