useChainlinkFunctions()

Collection of community submitted examples for Chainlink Functions

Get inspired from quick examples

Chainlink Functions, a new self-service platform that allows anyone to write serverless code to fetch any data from any API and run custom compute on Chainlink's network. Learn from examples created by the community or contribute your own!
Learn more about Chainlink Functions

Add Your Own Example

Fetch the UTC time

Submitted by:
Evan Drake

This function fetches the UTC timestamp from WorldTimeAPI.

```
const config = {
  url: "https://worldtimeapi.org/api/timezone/Etc/UTC",
};

const response = await Functions.makeHttpRequest(config);

const datetime = response.data.utc_datetime;

return Functions.encodeString(datetime);
```

Video or channel verification with an EVM wallet address

Submitted by:

Viet Nguyen

The function returns a result as uint256: 1 if the wallet is found in the video/channel description, and 0 if not found.

```

// Begin Function
// args = [videoOrChannelId, ownerWalletAddress, type]
const videoOrChannelId = args[0]; // video or channel id get from youtube eg. xyFa2amJ
const ownerWalletAddress = args[1]; // owner wallet address eg. 0x1282401445452436b409
const type = args[2]; // "video" | "channel"

// Youtube API key get from https://console.cloud.google.com/apis/dashboard
if (!secrets.apiKey) {
  throw Error(
    "YOUTUBE_API_KEY required"
  );
}

// Youtube API request
const youtubeRequest = Functions.makeHttpRequest({
  url: `https://youtube.googleapis.com/youtube/v3/${type}s`,
  method: "GET",
  params: {
    part: "snippet",
    id: videoOrChannelId,
    key: secrets.apiKey
  },
});

const youtubeResponse = await youtubeRequest;

if (youtubeResponse.error) {
  throw new Error("Youtube error");
}

// Checking youtube response if !youtubeResponse.data.items[0] -> Youtube video or cha
if (youtubeResponse.data && youtubeResponse.data.items && youtubeResponse.data.items[0
  const description = youtubeResponse.data.items[0].snippet.description.toLowerCase();
  const walletIndex = description.indexOf(ownerWalletAddress.toLowerCase());
  // If it found owner wallet address return 1, otherwise 0
  const resultInt = walletIndex !== -1 ? 1 : 0;
  return Functions.encodeUint256(resultInt);
} else {
  throw new Error("Youtube video or channel not found");
}
// End Function
```

Entity for an address from the Arkham API

Submitted by:

Arkham Team

This Function returns the entity ID of a given address. The entity ID is a string, and represents a slugified version of the entity's name. For example, Binance -> binance. The address is the only required parameter, and an API key is the only required secret.

```
// This Function returns the entity ID of the input address. Entity IDs are string
// slugified version of the entity's name. They can be used in other functions to Arkh
// you'd like to find more information about the entity and its on-chain footprint.

// An Arkham API key is required to use this function. Apply for one here: https://doc

const address = args[0]

// Validate address input.
if (address.length != 42) {
  throw Error("invalid address")
}

// Validate required secret.
if (!secrets.ARKHAM_API_KEY) {
  throw Error("api key required")
}

// Make the request to the /intelligence/address/:address/all endpoint.
const url = `https://api.arkhamintelligence.com/intelligence/address/${address}/all`
const resp = await Functions.makeHttpRequest({url, headers: {'API-Key': secrets.ARKHAM
const data = resp["data"]
if (resp.error) {
  console.error(error)
  throw Error("Request failed")
}

// Since we used the /all endpoint, we get data in the form of a chain map (see const
// In very rare cases, an address will have a different entity based on the chain. In
// you can choose which chain you'd like to privilege.
let entityId = ""
for (const [chain, intel] of Object.entries(data).sort()) { // Sort so that output is
  // Choose the chain with the first non-null arkhamEntity field.
  if (intel.arkhamEntity !== undefined) {
    entityId = intel.arkhamEntity.id
    break
  }
}

return Functions.encodeString(entityId)
```

Weight-Height results from Poke API

Submitted by:

Naman Gautam

This Function returns the Base Experience, Weight, Height of Pokemon. It uses the Poke API. Parameters includes name of pokemon.

```
// This function fetches Base Experience, Weight, Height of Pokemon results from P
// Args include name of pokemon

const pokiURL = "https://pokeapi.co/api/v2/pokemon"

const pokemonCharacter = args[0]

console.log(`Sending HTTP request to ${pokiURL}/${pokemonCharacter}/`)

const pokiRequest = Functions.makeHttpRequest({
url: `${pokiURL}/${pokemonCharacter}`,
method: "GET",
})

// Execute the API request (Promise)
const pokiResponse = await pokiRequest

if (pokiResponse.error) {
console.error(pokiResponse.error)
throw Error("Request failed, try checking the params provided")
}

console.log(pokiResponse)

// gets the Base Experience, Weight, Height of Pokemon
const reqData = pokiResponse.data

// Gives the whole response from the request
console.log(reqData)

// result is in JSON object, containing Base Experience, Weight, Height of Pokemon
const myData = {
base_experience: reqData.base_experience,
weight: reqData.weight/10, // The weight of this Pokemon in hectograms which is conver
height: reqData.height/10, // The height of this Pokemon in decimetres which is conver
}

// Use JSON.stringify() to convert from JSON object to JSON string
// Finally, use the helper Functions.encodeString() to encode from string to bytes
return Functions.encodeString(JSON.stringify(myData))
```

Latest News Headline from NEWS API

Submitted by:

Shikhar Agarwal

This Function returns the latest news headline for a particular country. It uses the NEWS API to get the news. Parameters include country code and keyword(if user want to filter search on the basis of keyword)

```
// This function fetches the latest news headline from a country
// Args include country code and keyword(if any)
// If user don't want to filter result on the basis of keyword, just pass an empty str

if (!secrets.apiKey) {
  throw Error("Api Key Not Found")
}

const url = "https://newsapi.org/v2/top-headlines?"

const country = args[0]          // Example - "in" for India
const keywordSearch = args[1]    // Example - "web3"  (This arg is optional, leave an

console.log(`Sending HTTP GET Request to ${url}country=${country}&q=${keywordSearch}`)

const newsRequest = Functions.makeHttpRequest({
  url: url,
  method: "GET",
  headers: {
    "X-Api-Key": secrets.apiKey
  },
  params: {
    country: country,
    q: keywordSearch
  }
})

// Execute the API request (Promise)
const newsResponse = await newsRequest

// check if there was any error during the request
if (newsResponse.error) {
  throw Error("Request failed")
}

// if there is no news, throw an error with the message
if (newsResponse.data.articles.length == 0) {
  throw Error("No news!")
}

// get the latest news
const newsSelect = newsResponse.data.articles[0]

// choosing the required parameters to be uploaded
const newsData = {
  publishTime: newsSelect.publishedAt,
  title: newsSelect.title
}
```

```

// Use JSON.stringify() to convert from JSON object to JSON string
// Finally, use the helper Functions.encodeString() to encode from string to bytes
return Functions.encodeString(JSON.stringify(newsData))
```

Asteroid Data from NASA API

Submitted by:

Naman Gautam

This Function returns the very first current close-approach data for asteroid in the given range of time and max distance

```
// This API provides access to current close-approach data for asteroid and comets in
// Args include des, date-min, date-max, dist-max
// des - only data for the object matching this designation (e.g., 2015 AB or 141P or
// date-min - exclude data earlier than this date YYYY-MM-DD
// date-max - exclude data later than this date YYYY-MM-DD
// dist-max - exclude data with an approach distance greater than this (in AU)


const sbdbURL = "https://ssd-api.jpl.nasa.gov/cad.api?"

const des = args[0]
const dateMin = args[1]
const dateMax = args[2]
const maxDist = args[3]

console.log(`Sending HTTP request to ${sbdbURL}des=${des}&date-min=${dateMin}&date-max

const sbdbRequest = Functions.makeHttpRequest({
  url: sbdbURL,
  method: "GET",
  params: {
    des: des,
    "date-min": dateMin,
    "date-max": dateMax,
    "dist-max": maxDist,
  },
})

// response from sbdb
const sbdbResponse = await sbdbRequest

if (sbdbResponse.error) {
  console.error(geoCodingResponse.error)
  throw Error("Request failed, try checking the params provided")
}

console.log(sbdbResponse)

// getting the very first data of an asteroid
const reqData = sbdbResponse.data.data[0]

// selecting the required output
const myData = {
  orbitId: reqData[1],
  closeTimeApproach: reqData[3],
  dist: reqData[4],
  relativeVelocity: reqData[7]
}
```

```

// Use JSON.stringify() to convert from JSON object to JSON string
// Finally, use the helper Functions.encodeString() to encode from string to bytes
return Functions.encodeString(JSON.stringify(myData))
```

Zerion wallet account $USD balance

Submitted by:

Benjamin

This function returns the USD balance of an account using the Zerion wallet tracker.

```
if (!secrets.zerionApiKey) {
    throw Error('API_KEY');
}

async function fetchBalanceFromZerion(address) {
    const config = {
        method: 'GET',
        headers: {
            accept: 'application/json',
            authorization: `Basic ${secrets.zerionApiKey}`
        },
        url: `https://api.zerion.io/v1/wallets/${address}/portfolio/?currency=usd`
    };

    const response = await Functions.makeHttpRequest(config);

    if (response.error) {
        throw new Error(response.response.data.message);
    }

    const upscaledUSDValue = response.data.data.attributes.total.positions * 10 ** 18;

    return upscaledUSDValue;
}

const address = args[0];
const balance = await fetchBalanceFromZerion(address);

return Functions.encodeUint256(balance);

// Gas usage can be reduced by using a minified version. Please remove the code above

// if(!secrets.zerionApiKey)throw Error("API_KEY");async function fetchBalanceFromZeri
```

Current Flight Status from Aviation Stack API

Submitted by:

Shikhar Agarwal

This Function returns the current flight status for a particular flight. It uses the aviation stack API to get the information of the flight. Parameters include airline iata and flight number

```
// This function fetches the latest flight status for a particular flight
// Args include the airline iata and flight number.
// Example - for indigo, airline iata is 6E

if (!secrets.apiKey) {
  throw Error("Aviation API Key is not available!")
}

// make HTTP request
const url = 'http://api.aviationstack.com/v1/flights?';
const airlineIata = args[0]  // example - "6E" airline iata for indigo
const flightNum = args[1]    // example - "123" flight number for indigo

console.log(`HTTP GET Request to ${url}airline_iata=${airlineIata}&flight_number=${fli


const flightrequest = Functions.makeHttpRequest({
    url: url,
    method: "GET",
    params: {
      airline_iata: airlineIata,
      flight_number: flightNum,
      access_key: secrets.apiKey
    },
})

// Execute the API request (Promise)
const flightResponse = await flightrequest

if (flightResponse.error) {
  throw Error("Request failed")
}

// to get the latest data for flight
const latestFlightData = flightResponse.data.data[0]
console.log(latestFlightData)

// bundle of all the required data in flightData object
const flightData = {
  date: latestFlightData.flight_date,
  status: latestFlightData.status,
  departureAirport: latestFlightData.departure.airport,
  departureTime: latestFlightData.departure.actual || latestFlightData.departure.estim
  arrivalAirport: latestFlightData.arrival.airport,
  arrivalTime: latestFlightData.arrival.actual || latestFlightData.arrival.estimated |
}

// Use JSON.stringify() to convert from JSON object to JSON string
```

```
// Finally, use the helper Functions.encodeString() to encode from string to bytes
return Functions.encodeString(JSON.stringify(flightData))
```

Current Temperature results from openweather API

Submitted by:

Shikhar Agarwal

This Function returns the current temperature in an area. It uses the openweather API. Parameters include zipcode and country code of the location, along with the apiKey in secrets, and units to get the temperature in Kelvin, Celsius or Fahrenheit

```
// This function fetches the latest temperature for a particular area from openweather
// Args include the zipcode of your location, ISO 3166 country code
// units- unit in which we want the temperature (standard, metric, imperial)


if (!secrets.apiKey) {
  throw Error("Weather API Key is not available!")
}

const zipCode = `${args[0]},${args[1]}`

const geoCodingURL = "http://api.openweathermap.org/geo/1.0/zip?"

console.log(`Sending HTTP request to ${geoCodingURL}zip=${zipCode}`)

const geoCodingRequest = Functions.makeHttpRequest({
    url: geoCodingURL,
    method: "GET",
    params: {
        zip: zipCode,
        appid: secrets.apiKey
    }
})

const geoCodingResponse = await geoCodingRequest;

if (geoCodingResponse.error) {
    console.error(geoCodingResponse.error)
    throw Error("Request failed, try checking the params provided")
}

console.log(geoCodingResponse);

const latitude = geoCodingResponse.data.lat
const longitude = geoCodingResponse.data.lon
const unit = args[2]

const url = `https://api.openweathermap.org/data/2.5/weather?`

console.log(`Sending HTTP request to ${url}lat=${latitude}&lon=${longitude}&units=${un

const weatherRequest = Functions.makeHttpRequest({
  url: url,
  method: "GET",
  params: {
    lat: latitude,
    lon: longitude,
    appid: secrets.apiKey,
```

```
    units: unit
  }
})

// Execute the API request (Promise)
const weatherResponse = await weatherRequest
if (weatherResponse.error) {
  console.error(weatherResponse.error)
  throw Error("Request failed, try checking the params provided")
}

// gets the current temperature
const temperature = weatherResponse.data.main.temp

// Gives the whole response from the request
console.log("Weather response", weatherResponse)

// result is in JSON object, containing only temperature
const result = {
  temp: temperature
}

// Use JSON.stringify() to convert from JSON object to JSON string
// Finally, use the helper Functions.encodeString() to encode from string to bytes
return Functions.encodeString(JSON.stringify(result))
```

Get US Treasury Yield of specified maturity and interval

Submitted by:

Justin Gnoh

This example shows how to return the daily, weekly, and monthly US treasury yield of a given maturity timeline from the Alphavantage API. Result is expected to be in percentage.

```
// This function retrieves the latest released yield of the US X Year Treasury fro
// Maturity timelines: 3month, 2year, 5year, 7year, 10year, 30year
// Interval options: daily, weekly, monthly
const maturity = args[0]
const interval = args[1]

if (!secrets.apiKey) {
  throw Error("Need to set Alpha Vantage API key");
}

// make HTTP request
const url = `https://www.alphavantage.co/query?function=TREASURY_YIELD`
console.log(`HTTP GET Request to ${url}&interval=${interval}&maturity=${maturity}`)

// construct the HTTP Request object. See: https://github.com/smartcontractkit/functio
// params used for URL query parameters
const alphavantageRequest = Functions.makeHttpRequest({
  url: url,
  params: {
    interval: interval,
    maturity: maturity,
    apikey: secrets.apiKey
  },
})

// Execute the API request (Promise)
const alphavantageResponse = await alphavantageRequest
if (alphavantageResponse.error) {
  console.error(alphavantageResponse.error)
  throw Error("Request failed")
}

const data = alphavantageResponse["data"]
console.log(data);
// Gets the latest yield rate in the array of returned data values
const floatingRate = data.data[0].value;

if (data.Response === "Error") {
  console.error(data.Message)
  throw Error(`Functional error. Read message: ${data.Message}`)
}

// Solidity doesn't support decimals so multiply by 100 and round to the nearest integ
// Use Functions.encodeUint256 to encode an unsigned integer to a Buffer
return Functions.encodeUint256(Math.round(floatingRate * 100))
```

Google Maps Distance Matrix API

Submitted by:

Karim Hadni

This function returns the distance and duration of a trip between two locations using the Google Maps Distance Matrix API. The origin and destination are required parameters. A Google Maps API key is also required and must be set as a secret.

```
// Define constants for the API endpoint and request parameters
const API_ENDPOINT = "https://maps.googleapis.com/maps/api/distancematrix/json"
const DEPARTURE_TIME = "now"
const RETURN_PROPERTIES = ["distance", "duration", "duration_in_traffic"]

// Get the arguments from the request config
const origin = args[0] // e.g. "New York City"
const destination = args[1] // e.g. "Washington DC"

// Get the Google Maps API Key from the environment variables
const apiKey = secrets.apiKey
if (
  !apiKey ||
  apiKey ===
    "Your Google Maps API key (get one: https://developers.google.com/maps/documentati
) {
  throw new Error("GOOGLE_MAPS_API_KEY environment variable not set or invalid")
}

// build HTTP request object
const requestParams = {
  url: `${API_ENDPOINT}?departure_time=${DEPARTURE_TIME}`,
  params: {
    origins: origin,
    destinations: destination,
    key: apiKey,
  },
}

// Make the HTTP request to the Google Maps API
const googleMapsRequest = Functions.makeHttpRequest(requestParams)
let response

try {
  response = await googleMapsRequest
} catch (error) {
  throw new Error(`Google Maps API request failed: ${error.message}`)
}

// Check if the response status is OK
if (response.status !== 200) {
  throw new Error(`Google Maps API returned an error: ${response.statusText}`)
}

// Extract the relevant data from the response
const data = response.data

// Check if the response contains the expected properties
```

```
if (!data.rows || !data.rows[0].elements || !data.rows[0].elements[0]) {
  throw new Error("Google Maps API response is missing expected data")
}

// Extract the distance, standard duration, and duration in traffic from the response
const distance = data.rows[0].elements[0].distance.value
const stdDuration = data.rows[0].elements[0].duration.value
const duration_in_traffic = data.rows[0].elements[0].duration_in_traffic.value

// Log the results for debugging purposes
console.log(`Distance: ${distance / 1000} km`)
console.log(`std duration: ${stdDuration / 60} min`)
console.log(`duration_in_traffic: ${duration_in_traffic / 60} min`)
console.log(`time in traffic jam: ${(duration_in_traffic - stdDuration) / 60} min`)

// Encode and return the distance (in meters), standard duration (in seconds), and dur
return Functions.encodeString(`${distance},${stdDuration},${duration_in_traffic}`)
```

Fetch Discord Upvote Data

Submitted by:

Sam Demaree

This function retrieves the number of upvotes a Discord member has received in the past 24 hours. *Note: ChatGPT was used to demonstrate that non-developers can also participate.

```
// This function retrieves the number of upvotes a Discord member has received in

const getDiscordUpvotes = async (memberId, apiKey, guildId, channelId, timeRangeMs) =>
    const endpoint = 'https://discord.com/api/v9'
    const timeRangeSec = Math.round(timeRangeMs / 1000)
    const time24HoursAgo = Math.round((Date.now() - timeRangeMs) / 1000)
    const headers = {
        'Authorization': `Bot ${apiKey}`,
        'Content-Type': 'application/json'
    }
    const config = {
        method: 'GET',
        headers: headers,
        url: `${endpoint}/guilds/${guildId}/audit-logs?limit=100&user_id=${memberId}&b
    }
    const response = await Functions.makeHttpRequest(config)
    if (response.error) {
        throw new Error(response.response.data.message)
    }
    const auditLogs = response.data.audit_log_entries
    let upvotes = 0
    for (let i = 0; i < auditLogs.length; i++) {
        const log = auditLogs[i]
        if (log.action_type === 72 && log.target_id === channelId && log.created_at >=
            upvotes++
        }
    }
    return Functions.encodeUint256(upvotes)
}
```

US election results from AP (Associated Press) API

Submitted by:

Karen Stepanyan

This Function returns the winner of the US election for a given date. It uses the AP (Associated Press) API to get the results. The date is the only required parameter. API

key is the only required secret.

```
// Chainlink Function to get election results from AP (Associated Press) API. Date and

const getReportingUnit = (reportingUnits, statePostal) => {
  const level = statePostal === 'US' ? 'national' : 'state'
  const reportingUnit = reportingUnits.find((ru) => ru.level === level)
  if (!reportingUnit) {
    throw new Error('Cannot find reporting unit')
  }
  return reportingUnit
}

const getReportingUnitWinner = (reportingUnit) => {
  for (const candidate of reportingUnit.candidates) {
    if (candidate.winner === 'X') {
      return candidate
    }
  }
  throw new Error('Candidate not found')
}


const date = args[0] // The date of the election formatted as YYYY-MM-DD
const statePostal = args[1] // The state's two-letter code e.g CO. `US` to get the res
const raceID = args[2] // AP-assigned race ID. Should be used with `statePostal`
const raceType = args[3] || 'G' // The race type the election is for. The race type ca
const resultsType = args[4] || 'L' // The type of results to return. `L` for live resu

if (!secrets.apikey) {
  throw new Error('Missing AP API key')
}

const params = {
  level: statePostal === 'US' ? 'national' : 'state',
  raceTypeID: raceType,
  format: 'json',
  winner: 'X',
  resultsType: resultsType,
  apikey: secrets.apikey,
}

if ((statePostal && !raceID) || (!statePostal && raceID)) {
  throw new Error('Both statePostal and raceID are required if one is provided')
}

if (statePostal) {
  params.statePostal = statePostal
}

```

```
if (raceID) {
  params.raceID = raceID
}

const config = {
  url: `https://api.ap.org/v3/elections/${date}`,
  params
}

const response = await Functions.makeHttpRequest(config)

const races = response.data.races
if (races.length === 0) {
  throw new Error('Could not find any races')
}
if (races.length > 1) {
  throw new Error('Finding the winner from multiple races is not supported')
}

const race = races[0]
const reportingUnit = getReportingUnit(race.reportingUnits, statePostal)
const raceWinner = getReportingUnitWinner(reportingUnit)


return Functions.encodeString(JSON.stringify(raceWinner))
```

Aggregate the ERC20 balance of an address across multiple chains

Submitted by:

polarzero

Find the balance of a user for a specific ERC20 token across the specified chains, and return the total balance. This balance, for example, could be used immediately in the callback function to approve or deny the user access to specific functions in the contract.

```
// https://github.com/polar0/cross-chain-ERC20-balance-verification/blob/main/implemen

// The address to check the balances of
const userAddress = args[0]
// The chains to check, formatted as:
// name:tokenAddress,name:tokenAddress...
const tokens = args[1].split(",").map((tokenAddress) => {
  const [chain, address] = tokenAddress.split(":")
  return { chain, address }
})

// Verify if there is indeed a secret (RPC URL) for each chain
tokens.forEach((token) => {
  if (!secrets[token.chain]) {
    throw new Error(`No secret found for chain ${token.chain}`)
  }
})

// Prepare requests for each chain
const requests = tokens.map((token, index) => {
  return Functions.makeHttpRequest({
    url: secrets[token.chain],
    method: "POST",
    data: {
      id: index,
      jsonrpc: "2.0",
      method: "eth_call",
      params: [
        {
          to: token.address,
          // The signature of 'balanceOf(address)' + the user address without the 0x p
          data: "0x70a082310000000000000000000000000" + userAddress.slice(2),
        },
        "latest",
      ],
    },
  })
})

// Wait for all requests to finish
const responses = await Promise.all(requests)

// Parse responses
const balances = responses.map((response) => {
  // Convert the result to a number
  return parseInt(response.data.result, 16) ?? 0
})

```

```
// Sum all balances
const totalBalance = balances.reduce((a, b) => a + b, 0)

// Return the total balance of the user
return Functions.encodeUint256(totalBalance)
```

Find the Best DEX Trade Value for a Given Asset Pair

Submitted by:

Max Melcher

This example shows how to return the best DEX trade value for a give asset pair using Paraswap DEX Aggregator

```
// Decimals can be passed from the token contract decimals() function
const srcToken = args[0] // Token source (selling)
const srcDecimals = args[1]
const destAsset = args[2] //Token destination (buying)
const destDecimals = args[3]
const amount = args[4] // Amount of source token to trade

// Pull from the Paraswap DEX Aggregator router
const paraswapRequest = await Functions.makeHttpRequest({
  url: `https://apiv5.paraswap.io/prices?srcToken=${srcToken}&srcDecimals=${srcDecimal
})

if (!paraswapRequest.error) {
  console.log("Optimal trade route found!")
  console.log(
    `Swap found to exchange ${
      10 ** -paraswapRequest.data.priceRoute.srcDecimals * parseInt(paraswapRequest.da
    } of ${paraswapRequest.data.priceRoute.srcToken} into ${
      10 ** -paraswapRequest.data.priceRoute.destDecimals * parseInt(paraswapRequest.d
    } of ${paraswapRequest.data.priceRoute.destToken}`
  )
  //Sample Output: "Swap found to exchange 1 of 0x514910771af9ca656af840dff83e8264ecf9
  console.log(`${paraswapRequest.data.priceRoute.bestRoute.length} best route(s) found
  //If direct swap is found with one pool return that pool address
  if (paraswapRequest.data.priceRoute.bestRoute[0].percent == 100) {
    console.log(
      `One direct route found through ${paraswapRequest.data.priceRoute.bestRoute[0].s
    )
    //Sample Output: One direct route found through UniswapV2
    console.log(paraswapRequest.data.priceRoute.bestRoute[0].swaps[0].swapExchanges[0]
    /*
    Sample Output:
    {
      router: '0xF9234CB08edb93c0d4a4d4c70cC3FfD070e78e07',
      path: [
        '0x514910771af9ca656af840dff83e8264ecf986ca',
        '0x6b175474e89094c44da98b954eedeac495271d0f'
      ],
      factory: '0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f',
      initCode: '0x96e8ac4277198ff8b6f785478aa9a39f403cb768dd02cbee326c3e7da348845f',
      feeFactor: 10000,
      pools: [
        {
          address: '0x6D4fd456eDecA58Cf53A8b586cd50754547DBDB2',
          fee: 30,
          direction: true
        }
      ],
```

```
        gasUSD: '2.735657'
      }
      */
    }
  } else {
    console.log("Paraswap Request error")
    console.log({ ...paraswapRequest })
  }
  return Functions.encodeUint256(parseInt(paraswapRequest.data.priceRoute.destAmount))
```

Fetch result of soccer match from Sportsdata.io

Submitted by:

Karen Stepanyan

The function fetches the result of soccer match. Required arguments are match date and abbreviations of team names

```
// Chainlink function to get the winner of soccer match. Possible return values are ab

const date = args[0] // Match date. basic date format YYYY-MM-DD. for example 2023-01-
let teams = args[1] //  competing teams in following format TEAM1/TEAM2. for example A

if (!secrets.soccerApiKey) {
  throw Error("Sportsdata.io API KEY is required")
}

const config = {
  url: `https://api.sportsdata.io/v3/soccer/scores/json/GamesByDate/${date}?key=${secr
}

const response = await Functions.makeHttpRequest(config)

const allMatches = response.data;

const match = allMatches.find(match => {
  const playingTeams = `${match.AwayTeamKey}/${match.HomeTeamKey}`.toUpperCase()
  const playingTeamsReversed = `${match.HomeTeamKey}/${match.AwayTeamKey}`.toUpperCase
  if (teams.toUpperCase() === playingTeams || teams.toUpperCase() === playingTeamsReve
    return true
  }
})

if (!match) {
  throw new Error('Match not found for given arguments')
}

if (match.Winner === 'Scrambled') {
  throw new Error('Data is scrambled, use production API Key')
}

let result;

if (match.Winner === 'AwayTeam') {
  result = match.AwayTeamKey
} else if (match.Winner === 'HomeTeam') {
  result = match.HomeTeamKey
} else if (match.Winner === 'Draw') {
  result = 'Draw'
}

if (!result) {
  throw new Error('Could not get the winner team.')
}

return Functions.encodeString(result)
```

Prompt AI for a response

Submitted by:

Patrick Collins

Ask OpenAI (or any AI model you want to interact with) for information on-chain.

```
const prompt = args[0]

if (
    !secrets.openaiKey
) {
    throw Error(
        "Need to set OPENAI_KEY environment variable"
    )
}

// example request:
// curl https://api.openai.com/v1/completions -H "Content-Type: application/json" -H "

// example response:
// {"id":"cmpl-6jFdLbY08kJobPRfCZL4SVzQ6eidJ","object":"text_completion","created":167
const openAIRequest = Functions.makeHttpRequest({
    url: "https://api.openai.com/v1/completions",
    method: "POST",
    headers: {
        'Authorization': `Bearer ${secrets.openaiKey}`
    },
    data: { "model": "text-davinci-003", "prompt": prompt, "temperature": 0, "max_toke
})

const [openAiResponse] = await Promise.all([
    openAIRequest
])
console.log("raw response", openAiResponse)

const result = openAiResponse.data.choices[0].text
return Functions.encodeString(result)
```

Read cross-chain information

Submitted by:

Patrick Collins

The function reads the supply APY rate of depositing WETH into AaveV3 on Polygon

```
// This example shows how to make a decentralized price feed using multiple APIs

// Arguments can be provided when a request is initated on-chain and used in the reque
const contractAddress = args[0]
const encodedAbiFunctionCall = args[1]

if (
    !secrets.polygonKey
) {
    throw Error(
        "Need to set POLYGON_RPC_URL environment variable"
    )
}

// curl --data '{"method":"eth_call","params":[{"to":"0x794a61358D6845594F94dc1DB02A25
// example response:
// {"jsonrpc":"2.0","id":1,"result":"0x0000000000000000000000003e8000006914000039cb03e

// To make an HTTP request, use the Functions.makeHttpRequest function
// Functions.makeHttpRequest function parameters:
// - url
// - method (optional, defaults to 'GET')
// - headers: headers supplied as an object (optional)
// - params: URL query parameters supplied as an object (optional)
// - data: request body supplied as an object (optional)
// - timeout: maximum request duration in ms (optional, defaults to 10000ms)
// - responseType: expected response type (optional, defaults to 'json')

// Ideally, you'd use multiple RPC URLs so we don't have to trust just one
const polygonReadRequest = Functions.makeHttpRequest({
    url: secrets.polygonKey,
    method: "POST",
    data: {
        "jsonrpc": "2.0",
        "method": "eth_call",
        "params": [
            { "to": contractAddress, data: encodedAbiFunctionCall },
            "latest"
        ],
        "id": 1
    }
})

// First, execute all the API requests are executed concurrently, then wait for the re
const [polygonResponse] = await Promise.all([
    polygonReadRequest
])

```

```
console.log("raw response", polygonResponse)

// take the "0x" off the front of the hex string
const result = polygonResponse.data.result.slice(2)

// loop through result and convert each 64 characters to a number
const startingIndex = 64 * 2
const supplyApy = "0x" + result.slice(startingIndex, startingIndex + 64)

// convert the hex supplyApy to a number
const supplyApyNumber = parseInt(supplyApy, 16)
// This number is returned as a RAY, so we'd divide by 1e27, or 1e25 to get a percenta
console.log("WETH Supply APY on AaveV3 in Polygon: ", (supplyApyNumber / 1e25), "%")

// The source code MUST return a Buffer or the request will return an error message
// Use one of the following functions to convert to a Buffer representing the response
// - Functions.encodeUint256
// - Functions.encodeInt256
// - Functions.encodeString
// Or return a custom Buffer for a custom byte encoding
// return Functions.encodeUint256(Math.round(medianPrice * 100))
return Functions.encodeUint256(supplyApyNumber)
```

Fetch outcome of off-chain Snapshot.org vote

Submitted by:

ChainLinkGod

The function fetches the outcome of an off-chain Snapshot.org vote proposal using the GraphQL API. Takes into account if the vote has closed and has met quorum. Gas efficient solution for DAOs.

```
const proposalID = args[0]

if (!proposalID) {
  throw Error("Proposal ID is required")
}

const config = {
  url: "https://hub.snapshot.org/graphql?",
  method: "POST",
  headers: {
    'content-type': 'application/json'
  },
  params: {
    operationName: "Proposal",
    query: `query Proposal {\n  proposal(id:"${proposalID}") {\n    id\n    votes\n
    variables: null,
  },
}

const response = await Functions.makeHttpRequest(config)

const state = response.data.data.proposal.state
const totalScore = response.data.data.proposal.scores_total
const quorum = response.data.data.proposal.quorum

if (state !== 'closed') {
  return Functions.encodeString('Vote not ended')
}

if (totalScore < quorum) {
  return Functions.encodeString('Quorum not met')
}

const scores = response.data.data.proposal.scores
const choices = response.data.data.proposal.choices
const highestIndex = scores.indexOf(Math.max(...scores));

return Functions.encodeString(choices[highestIndex])
```

Financial metric data for dApps and blockchains sourced from Token Terminal

Submitted by:

ChainLinkGod

This Function fetches metric data from the Token Terminal API for a specific project. Supported metrics include revenue, fees, earnings, active users, TVL, volume, supply, and more. Projects includes both dApps and blockchains. Optional parameter for specific date. Requires Token Terminal Pro subscription to obtain API key.

```

const metric = args[0] // valid metric id that can be found on https://api.tokentermin
const project = args[1] // project id
const date = args[2] // optional date. format YYYY-MM-DD. For example 2023-02-10
const apiKey = secrets.API_KEY;


if (!apiKey) {
  throw Error("Tokenterminal API Key is required")
}

const config = {
  url: `https://api.tokenterminal.com/v2/metrics/${metric}?project_ids=${project}`,
  headers: {
    'Authorization': `Bearer ${apiKey}`
  }
}

const response = await Functions.makeHttpRequest(config)
if (response.error) {
  throw new Error(response.response.data.message)
}

let data;
if (date) {
  data = response.data.data.find(d => d.timestamp.includes(date))
}else {
  data = response.data.data[0]
}
const result = Math.round(data.value * 100)

return Functions.encodeUint256(result)
```

Obtain outcome of off-chain vote

Submitted by:

mykcryptodev

This function fetches the final outcome of an off-chain vote on the Snapshot.org platform

```
const proposalId = args[0]

// Use snapshot's graphql API to get the final vote outcome
const snapshotRequest = () => Functions.makeHttpRequest({
  url: `https://hub.snapshot.org/graphql`,
  method: "POST",
  data: {
    query: `{
      proposal(id: "${proposalId}") {
        choices
        scores
        scores_state
      }
    }`,
  },
})

const { data, error } = await snapshotRequest()

if (error) {
  throw Error("Snapshot request failed")
}

const { proposal } = data.data
const { choices, scores, scores_state } = proposal

if (scores_state !== "final") {
  throw Error("Snapshot vote is not final")
}

const winningChoice = choices[scores.indexOf(Math.max(...scores))]
return Functions.encodeString(winningChoice)
```

Fetch and return available balance of Stripe account

Submitted by:

Karen Stepanyan

This function will fetch Stripe account available balance of particular currency.

```
const apiKey = secrets.API_KEY
const balanceCurrency = args[0] || 'usd'

if (!apiKey) {
  throw Error("Stripe API Key is required")
}


const config = {
  url: `https://${apiKey}@api.stripe.com/v1/balance`,
}

const response = await Functions.makeHttpRequest(config)

const balance = response.data.available.find(c => c.currency.toLowerCase() === balance

const balanceInCents = Math.round(balance.amount * 100)

return Functions.encodeUint256(balanceInCents)
```

Calculate the median price of a token on Uniswap V2

Submitted by:

moonthoon

This function calculates the median price of a token that is on Uniswap V2. It works by sampling up to 4 prices over a given time period then chooses the median value

```
// Max sample size is 4 due to 5 http request limit
const SAMPLE_SIZE = 4
// The number of decimals the price in USD is formatted to
const DECIMALS = 18
// A block buffer to take into consideration the synchronization of the subgraph
const GRAPH_BLOCK_BUFFER = 50
const AVG_SECONDS_PER_BLOCK = 12

// Token address
const token = args[0].toLowerCase();
// Pair address
const pair = args[1]
// Period in seconds
const period = args[2]

const blockRange = period / AVG_SECONDS_PER_BLOCK

if (!secrets.rpc) {
  throw Error("\"rpc\" environment variable not set")
}

const blockNumberResponse = await Functions.makeHttpRequest({
  url: secrets.rpc,
  method: "POST",
  headers: {
    "Accept": "application/json",
    "Content-Type": "application/json",
  },
  data: JSON.stringify({
    jsonrpc: "2.0",
    method: "eth_blockNumber",
    params: [],
    id: "1",
  }),
})

if (blockNumberResponse.error) {
  throw Error("Unable to fetch current block number")
}

const blockNumber = parseInt(blockNumberResponse.data.result, 16) - GRAPH_BLOCK_BUFFE

const fetchPrice = (blockNumber) => Functions.makeHttpRequest({
  url: "https://api.thegraph.com/subgraphs/name/uniswap/uniswap-v2",
  method: "POST",
  data: {
    query: `{
      pair(id: "${pair}", block: {number: ${blockNumber}}) {
```

```
          token0 {
            id
          }
          token1 {
            id
          }
          reserve0
          reserve1
          reserveUSD
        }
      }`,
    },
  })

  const stringToBigInt = (str) => {
    const splitStr = str.split(".")
    const decimals = splitStr[1].slice(0, DECIMALS).padEnd(DECIMALS, "0")
    return BigInt(`${splitStr[0]}${decimals}`)
  }

  const getPrice = async (blockNumber) => {
    const {
      error,
        data: {
          errors,
          data,
        },
    } = await fetchPrice(blockNumber)
    if (error.error || errors) {
      throw Error("Unable to fetch price from subgraph")
    }
    const { pair: { token0: { id: token0 }, token1: { id: token1 }, reserve0, reserve1,
    const token0LC = token0.toLowerCase()
    const token1LC = token1.toLowerCase()
    if (token0LC !== token && token1LC !== token) {
      throw Error("Token not found as part of the pair")
    }
    const tokenReserveInUSD = stringToBigInt(reserveUSD) / 2n
    const tokenReserve = stringToBigInt(token0LC === token ? reserve0 : reserve1)
    return BigInt(10 ** DECIMALS) * tokenReserveInUSD / tokenReserve
  }

  const pickRandomBlock = () => {
    return blockNumber - Math.round(Math.random() * blockRange)
  }

  let prices = []
  for (let i = 0; i < SAMPLE_SIZE; i++) {
```

```
    const price = await getPrice(pickRandomBlock())
    prices.push(price)
  }

  const midpoint = SAMPLE_SIZE % 2 === 0 ? SAMPLE_SIZE / 2 : (SAMPLE_SIZE + 1) / 2
  const median = prices[midpoint]

  return Functions.encodeUint256(median)
```

Twitter account verification with an Ethereum address

Submitted by:

polarzero

Check if a Twitter account belongs to a specific Ethereum address. This example uses the Twitter API to retrieve a user's recent tweets, and checks if they tweeted a specific message containing their address. It provides the arguments and returns the result via Chainlink Functions, which allows for prior validation of the user's ownership of the address via a signature or other method, thus performing a secure and non-intrusive verification.

```
// https://github.com/polar0/twitter-verifier-chainlink-functions/blob/main/implementa

// Get the arguments from the request config
const twitterUsername = args[0]; // e.g. 'TwitterDev'
const ethereumAddress = args[1]; // e.g. '0x1234567890123456789012345678901234567890'
// The string that must be included in the latest tweets of the user for the verificat
const requiredStringIncluded = `Verifying my Twitter account for ${ethereumAddress}`;
// How many tweets to check (min 5, max 100)
const MAX_RESULTS = 10;

// Initialize the result to -1 (error)
let result = -1;

// Get the bearer token from the environment variables
if (!secrets.apiKey) {
  throw Error(
    'TWITTER_BEARER_TOKEN environment variable not set for Twitter API. Get a free one
  );
}

// Don't even try if the username or address is empty
if (!twitterUsername || !ethereumAddress) {
  throw Error('Twitter username or Ethereum address is empty');
}

// Prepare the API requests
const twitterRequest = {
  // Get the user id from the provided username
  userIdByUsername: () =>
    Functions.makeHttpRequest({
      url: `https://api.twitter.com/2/users/by/username/${twitterUsername}`,
      headers: { Authorization: `Bearer ${secrets.apiKey}` },
    }),
  // Get the latest n tweets from the user (n = MAX_RESULTS)
  lastTweetsByUserId: (userId) =>
    Functions.makeHttpRequest({
      url: `https://api.twitter.com/2/users/${userId}/tweets?max_results=${MAX_RESULTS
      headers: { Authorization: `Bearer ${secrets.apiKey}` },
    }),
};

// First, request the user id from their username
const idRes = await new Promise((resolve, reject) => {
  twitterRequest.userIdByUsername().then((res) => {
    if (!res.error) {
      resolve(res);
    } else {
      reject(res);
```

```
    }
  });
});

if (idRes.error) {
  throw Error('Twitter API request failed - coult not get user id');
}

// Grab the user id
const userId = idRes.data.data.id || null;

// Let's be extra careful and make sure the user id is not null
if (!userId) {
  throw Error('Twitter API request failed - user id is null');
}

// Then, request the latest tweets
const tweetsRes = await new Promise((resolve, reject) => {
  twitterRequest.lastTweetsByUserId(userId).then((res) => {
    if (!res.error) {
      resolve(res);
    } else {
      reject(res);
    }
  });
});

if (tweetsRes.error) {
  throw Error('Twitter API request failed - coult not get tweets');
}

// It'll only get here if the request was successful
const tweets = tweetsRes.data.data;
const tweetTexts = tweets.map((tweet) => tweet.text);
// Check if any of these tweets include the required string
const res = tweetTexts.some((text) =>
  text.toLowerCase().includes(requiredStringIncluded.toLowerCase()),
);
// If it found the string, return 1, otherwise 0
result = res ? 1 : 0;

// `result` can either be:
// - 1 (verified)
// - 0 (not verified)
// - -1 (if by any chance no error was thrown, yet it could not verify)

// Return the result along with the username and address, which can be parsed and spli
return Functions.encodeString(
```

```
  `${result},${twitterUsername},${ethereumAddress}`,
);

```

Price data from multiple sources

Submitted by:

Morgan Kuphal

Retrieve the price of an asset from multiple API sources. Assets could be practially anything, incuding equities, crypto, or commodities. This example pulles from multiple different data providers (APIs) and derrives the median to return on chain via Chainlink Functions.

```
const coinMarketCapCoinId = args[0];
const coinGeckoCoinId = args[1];
const coinPaprikaCoinId = args[2];
const badApiCoinId = args[3];

const scalingFactor = parseInt(args[4]);

if (!secrets.apiKey) {
  throw Error('API_KEY environment variable not set for CoinMarketCap API.  Get a free
}

// OCR2DR.makeHttpRequest function parameters:
// - url
// - method (optional, defaults to 'GET')
// - headers: headers supplied as an object (optional)
// - params: URL query parameters supplied as an object (optional)
// - data: request body supplied as an object (optional)
// - timeout: maximum request duration in ms (optional, defaults to 10000ms)
// - responseType: expected response type (optional, defaults to 'json')

// Use multiple APIs & aggregate the results to enhance decentralization
const coinMarketCapResponse = await OCR2DR.makeHttpRequest({
  url: `https://pro-api.coinmarketcap.com/v1/cryptocurrency/quotes/latest?convert=USD&
  // Get a free API key from https://coinmarketcap.com/api/
  headers: { 'X-CMC_PRO_API_KEY': secrets.apiKey }
});
const coinGeckoResponse = await OCR2DR.makeHttpRequest({
  url: `https://api.coingecko.com/api/v3/simple/price?ids=${coinGeckoCoinId}&vs_curren
});
const coinPaprikaResponse = await OCR2DR.makeHttpRequest({
  url: `https://api.coinpaprika.com/v1/tickers/${coinPaprikaCoinId}`
});
const badApiResponse = await OCR2DR.makeHttpRequest({
  url: `https://badapi.com/price/symbol/${badApiCoinId}`
});

const prices = [];

if (!coinMarketCapResponse.error) {
  prices.push(coinMarketCapResponse.data.data[coinMarketCapCoinId].quote.USD.price);
}
else {
  console.log('CoinMarketCap Error');
  console.log({ ...coinMarketCapResponse });
}
if (!coinGeckoResponse.error) {
  prices.push(coinGeckoResponse.data[coinGeckoCoinId].usd);
} else {
```

```
    console.log('CoinGecko Error');
    console.log({ ...coinGeckoResponse });
  }
  if (!coinPaprikaResponse.error) {
    prices.push(coinPaprikaResponse.data.quotes.USD.price);
  } else {
    console.log('CoinPaprika Error');
    console.log({ ...coinPaprikaResponse });
  }

  // A single failed API request does not cause the whole request to fail
  if (!badApiResponse.error) {
    prices.push(httpResponses[3].data.price.usd);
  } else {
    console.log('Bad API request failed. (This message is expected and just for demonstr
  }

  // At least 3 prices are needed to aggregate the median price
  if (prices.length < 3) {
    // If an error is thrown, it will be returned back to the smart contract
    throw Error('More than 1 API failed');
  }

  const medianPrice = prices.sort((a, b) => a - b)[Math.round(prices.length / 2)];
  console.log(`Median Bitcoin price: $${medianPrice.toFixed(2)}`);

  // Use the following functions to encode a single value:
  // - OCR2DR.encodeUint256
  // - OCR2DR.encodeInt256
  // - OCR2DR.encodeString
  // Or return a Buffer for a custom byte encoding
  return OCR2DR.encodeUint256(Math.round(medianPrice * 100));
```

What will you build with Chainlink Functions?

Add Your Own Example

Read Chainlink Docs

Code examples on this site are contributed by the community and have not been audited. Use at your own risk. This site was a weekend hack project built to inspire the community.