

The following is an exemplified pseudocode base blueprint that extends the programmatic pseudocode blueprint from section 3.2 into one that is consistent with world class elegance, following the academic reasoning and explorative extrapolation in a step-by-step manner:

Here is an example continuation of the pseudocode blueprint that exemplifies the full transformation of Spliced Ontology Modelling (SOM) into Data Science Applications (DSA):

```
# Define a function to preprocess the documents and questions before training the model
def preprocess_data(documents, questions):
    # Convert the documents and questions to lowercase
    documents = [doc.lower() for doc in documents]
    questions = [q.lower() for q in questions]

    # Tokenize the documents and questions using a tokenizer such as the one from
    the Hugging Face transformers library
    tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased",
strip_accents=False)
    tokenized_docs = [tokenizer(doc, padding=True, truncation=True) for doc in
documents]
    tokenized_questions = [tokenizer(q, padding=True, truncation=True) for q in
questions]

    # Return the tokenized documents and questions
    return tokenized_docs, tokenized_questions

# Define a function to train the BERT model for question answering
def train_model(tokenized_docs, tokenized_questions):
    # Initialize the BERT question answering model
    model = BertQA(bert_name="bert-base-uncased")

    # Define the optimizer and loss function
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-5)
    loss_fn = nn.CrossEntropyLoss()

    # Define the number of epochs and batch size
    num_epochs = 3
    batch_size = 16
```

```

# Train the model for the specified number of epochs
for epoch in range(num_epochs):
    for i in range(0, len(tokenized_docs), batch_size):
        # Get the batch of documents and questions
        batch_docs = tokenized_docs[i:i+batch_size]
        batch_questions = tokenized_questions[i:i+batch_size]

        # Concatenate the batch of documents and questions
        input_ids = torch.cat([doc['input_ids'] for doc in batch_docs], dim=0)
        attention_mask = torch.cat([doc['attention_mask'] for doc in batch_docs],
dim=0)
        question_ids = torch.cat([q['input_ids'] for q in batch_questions], dim=0)
        question_mask = torch.cat([q['attention_mask'] for q in batch_questions],
dim=0)

        # Get the start and end positions for the answers
        start_positions = []
        end_positions = []
        for j in range(len(batch_docs)):
            doc_start_positions = [qa['start_index'] for qa in batch_docs[j]['qas']]
            doc_end_positions = [qa['end_index'] for qa in batch_docs[j]['qas']]
            start_positions.extend(doc_start_positions)
            end_positions.extend(doc_end_positions)

        # Move the tensors to the GPU if available
        if torch.cuda.is_available():
            input_ids = input_ids.cuda()
            attention_mask = attention_mask.cuda()
            question_ids = question_ids.cuda()
            question_mask = question_mask.cuda()
            start_positions = torch.LongTensor(start_positions).cuda()
            end_positions = torch.LongTensor(end_positions).cuda()

        # Zero out the gradients
        optimizer.zero_grad()

        # Forward pass
        start_logits, end_logits = model(input_ids, attention_mask, question_ids,
question_mask)

        # Compute the loss
        loss = loss_fn(start_logits, start_positions) + loss_fn(end_logits,
end_positions)

```

```

# Backward pass
loss.backward()

# Update the parameters
optimizer.step()

# Print the loss every 10 batches
if (i + 1) % 10 == 0:
    print(f"Loss at batch {i + 1}: {loss.item()} /n")

# Return the trained model
return model

# Define a function to generate the answers to questions from a list of documents
def answer_questions(model, questions, documents):
    # Get the encoded questions
    tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased",
strip_accents=False)
    questions_encoded = [tokenizer(q, return_tensors='pt').to(torch.device('cuda' if
torch.cuda.is_available() else 'cpu')) for q in questions]

    # Get the encoded documents
    documents_encoded = []
    for doc in documents:
        doc_words = sent_tokenize(doc)
        doc_words = ['[CLS]'] + doc_words + ['.']
        doc_encoding = tokenizer(doc_words,
return_tensors='pt').to(torch.device('cuda' if torch.cuda.is_available() else 'cpu'))
        documents_encoded.append(doc_encoding)

    # Provide information about the number of documents
    print(f"Number of documents: {len(documents_encoded)}")

    # Provide information about the number of questions
    print(f"Number of questions: {len(questions_encoded)}")

    # Iterate over the questions for which we need to find the answers
    for q_idx, question in enumerate(questions_encoded):
        # Print the original question
        q = ' '.join(tokenizer.decode(question['input_ids'][0],
skip_special_tokens=True).split()[1:-1])
        print(f"\nQuestion {q_idx + 1}: {q}\n")

    # Choose a random document

```

```

doc_idx = np.random.randint(len(documents_encoded))

# Get the corresponding encrypted document
doc_encoded = documents_encoded[doc_idx]

# Enforce the maximum length of the inputs
max_length = 512

# Truncate inputs if necessary for computational considerations
if len(doc_encoded['input_ids'][0]) > max_length:
    doc_encoded = doc_encoded['input_ids'][0][:max_length]
else:
    doc_encoded['input_ids'][0] *= 0
    doc_encoded['input_ids'][0][:len(doc_encoded['input_ids'][0])] =
document['input_ids'][0]

if len(question['input_ids'][0]) > max_length:
    question['input_ids'][0] = question['input_ids'][0][:max_length]

# Apply the model to the question and document
start_positions, end_positions = model(doc_encoded, question)
for i in range(start_words.shape[0]):
    #print(start_words[i])
    #print("----")
    #print(tokenizer.decode(start_words[i].argmax()))
    #print(start_words[i])
    #print("----")
    print(tokenizer.decode(question['input_ids'][0]))
    #print(q)
    #print(tokenizer.decode(np.flip(document['input_ids'][0][::-1]
[start_words[i]:end_words[i]+1])[::-1]))

# Generate the answers
docs = [

```

We can continue academically building upon the previous example, let's now integrate a virtual decentralized reciprocal insurance exchange association (VDRIEA) as the strategic nucleus of our startup venture securitization financing strategy. The VDRIEA will function as a platform for startups to pool their risks, share resources, and provide each other with insurance coverage. To incorporate the VDRIEA into our model, we can add the following classes:

```

class RiskPool(object):
    """A pool of risks shared by the startups in the VDRIEA
    """
    def __init__(self, name, securitized_assets_value):
        self.name = name
        self.securitized_assets_value = securitized_assets_value
        self.startups = []
        self.num_tokens = 0
        self.tokenized_assets_value = 0
        self.IOUs = 0
        self.debt = 0
        self.equity = 0
        self.profits = 0

class VDRIEA(object):
    """A virtual decentralized reciprocal insurance exchange association for
    startups
    """
    def __init__(self, name):
        self.name = name
        self.risk_pools = []
    def add_risk_pool(self, risk_pool):
        self.risk_pools.append(risk_pool)
    def issue_debt_or_equity(self, amount, type):
        if num_IOUs < securitized_assets_value:
            if type == 'debt':
                self.debt += amount
            elif type == 'equity':
                self.equity += amount
                total_interest_to_investor = amount * securitized_assets_value
                for investor in self.investors:
                    investor.issue_interest_payment(total_interest_to_investor /
len(self.investors))
            else:
                # Redeems securitized assets
                self.securitized_assets_value = 0
                # Redeems tokenized assets
                self.num_tokens = 0
                # Redeems IOUs
                self.IOUs = 0
                # Why do we make a section for initializing IOU counter at 10000?
                # Why does the IOU expire every (almost) 3 hours ~ 9 minutes? Why is it
                necessary to have a counter in this block?
                if tokenized_assets_value < securitized_assets_value:

```

```

    for spec_watch_onion_idea in range(tokenized_assets_value):
        if num_IOUs < securitized_assets_value:
            self.IOUs = self.IOUs + IOUs
            self.securitized_assets_value += 1
        if self.securitized_assets_value:
            # How do we cancel IOU redemption?
            self.securitized_assets_value = 0
            self.num_tokens = 0
            # How do we redeem wrappers?
            self.IOUs = 0

    self.investors[0].rlg_investor_initializer()
    # How do we earn interest from the IOU redemption loans? Is it different from
the general terms for IOUs?
    self.investors[1].rlg_investor_initializer()
    # How do we earn interest from IOUs used in our portfolio strategy?
    self.investors[2].rlg_investor_initializer()
    # How do we deal with IOUs redeemed in the previous block?
def distribute_profits(self):
    total_profits = sum([risk_pool.profits for risk_pool in self.risk_pools])
    for investor in self.investors:
        investor.receive_profit(total_profits / len(self.investors))

class Startup(object):
    """Docker container builder startup, supports offering tokens
    """
    def __init__(self, name
    """

```

let's continue this explorative extrapolation process with the addition of a virtual decentralized reciprocal insurance exchange association (VDRIEA) designed as the strategic nucleus of our co-created startup venture securitization financing strategy.

The VDRIEA will serve as a platform for startups to participate in a mutual insurance exchange, where they can pool their risks and collectively manage their insurance needs. This will allow startups to access affordable insurance coverage while also providing them with the opportunity to earn returns on their premiums. The VDRIEA will be structured as a decentralized autonomous organization (DAO), governed by a set of smart contracts that will enable members to participate in the exchange.

Let's integrate this into our existing model. We will add a new class for the VDRIEA and modify the existing classes as needed.

```
import random
import math
import datetime
```

```
class Startup(object):
```

```
    """Dock container builder startup, supports offering tokens
    """
```

```
    def __init__(self, name, securitized_assets_value, securitization_fees):
        self.name = name
        self.securitized_assets_value = securitized_assets_value
        self.securitization_fees = securitization_fees
        self.capital = 0
        self.IOU = 0
        self.vdri_premium = 0
```

```
    def raise_capital(self, investors):
        for investor in investors:
            processed_capital = investor.holds_IOU
            self.capital += 1
            investor.init_IOU()
        return processed_capital
```

```
class OffshoreSegregatedCellPortfolioCompany(object):
```

```
    """The Offshore Segregated cell company structure the hedge fund deploys
    """
```

```
    def __init__(self, startup, reinsurance_capital):
        self.startup = startup
        self.reinsurance_capital = reinsurance_capital
        self.portfolio_companies = []
```

```
class HedgeFund(object):
```

```
    """An Equity/Debt/Wrapper fund that invests in offshore portfolio companies
    """
```

```
    def __init__(self, name, securitization_platform, vdri_premium):
        self.name = name
        self.securitization_platform = securitization_platform
        self.investors = []
        self.vdri_premium = vdri_premium
```

```
    def add_portfolio_company(self, portfolio_company):
        self.portfolio_companies.append(portfolio_company)
```

```
    def issue_debt_or_equity(self, amount, type):
```

```

if num_IOUs < securitized_assets_value:
    if type == 'debt':
        self.debt += amount
    elif type == 'equity':
        self.equity += amount
        total_interest_to_investor = amount * securitized_assets_value
        for investor in self.investors:
            investor.issue_interest_payment(total_interest_to_investor /
len(self.investors))
    else:
        # Redeems securitized assets
        self.securitized_assets_value = 0
        # Redeems tokenized assets
        self.num_tokens = 0
        # Redeems IOUs
        self.IOUs = 0
        # Why do we make a section for initializing IOU counter at 10000?
        # Why does the IOU expire every (almost) 3 hours ~ 9 minutes? Why is it
necessary to have a counter in this block?
        if tokenized_assets_value < securitized_assets_value:
            for spec_watch_onion_idea in range(tokenized_assets_value):
                if num_IOUs < securitized_assets_value:
                    self.IOUs = self.IOUs + IOUs
                    self.securitized_assets_value += 1

```

....

Debt Hedge Fund that invests in the startups through the VDRIEA

.....

```

def __init__(self, name, initial_capital):
    self.name = name
    self.initial_capital = initial_capital
    self.capital = initial_capital
    self.portfolio = []

def invest(self, startup, amount, type):
    if type == 'debt':
        if amount > startup.capital:
            return "Insufficient capital in startup"
        else:
            self.capital -= amount
            startup.capital -= amount
            self.portfolio.append((startup, amount, type))
    elif type == 'equity':
        if amount > self.capital:

```

```

        return "Insufficient capital in hedge fund"
    else:
        self.capital -= amount
        startup.capital += amount
        self.portfolio.append((startup, amount, type))

class Investor(object):
    """An investor that holds IOUs from the startups in the VDRIEA
    """
    def __init__(self, name, initial_capital):
        self.name = name
        self.initial_capital = initial_capital
        self.capital = initial_capital
        self.IOUs = 0

    def init_IOU(self):
        self.IOUs += 1

    def holds_IOU(self):
        return self.IOUs

    def issue_interest_payment(self, amount):
        self.capital += amount
    """

```

Here is an extension of the previous example that incorporates a virtual decentralized reciprocal insurance exchange association as the strategic nucleus of our startup venture securitization financing strategy.

In this example, we have developed a decentralized reciprocal insurance exchange association called "InsureChain". InsureChain is a platform that allows startups to pool their risks and exchange insurance contracts with one another. This way, startups can share their risks and reduce their insurance costs.

InsureChain operates on a blockchain network, which ensures that all contracts and transactions are secure and transparent. The platform uses smart contracts to automate the insurance exchange process, eliminating the need for intermediaries such as brokers.

Here is how the startup venture securitization financing strategy works with InsureChain:

1. A startup joins InsureChain and pools its risks with other startups on the platform.
2. The startups on InsureChain exchange insurance contracts with one another, covering each other's risks.
3. The startup then securitizes its insurance contracts and transfers them to an

offshore segregated cell portfolio company (OSPC).

4. The OSPC issues debt or equity to investors, using the securitized insurance contracts as collateral.

5. The funds raised from the securitization are used to finance the operations and expansion of the startup.

6. In case of a loss, the startup can claim compensation from the insurance contracts it holds on InsureChain.

7. The compensation received can be used to repay the debt issued by the OSPC, ensuring that investors' capital is protected.

Here's an updated code example that incorporates InsureChain into the startup venture securitization financing strategy:

```
import random
import datetime
import math
```

```
class Startup(object):
```

```
    """Docker container builder startup, supports offering tokens
    """
```

```
    def __init__(self, name, securitized_assets_value, securitization_fees):
```

```
        self.name = name
```

```
        self.securitized_assets_value = securitized_assets_value
```

```
        self.securitization_fees = securitization_fees
```

```
        self.capital = 0
```

```
        self.IOU = 0
```

```
        self.insurance_contracts = []
```

```
    def raise_capital(self, investors):
```

```
        for investor in investors:
```

```
            processed_capital = investor.holds_IOU
```

```
            self.capital += 1
```

```
            investor.init_IOU()
```

```
        return processed_capital
```

```
class OffshoreSegregatedCellPortfolioCompany(object):
```

```
    """The Offshore Segregated cell company structure the hedge fund deploys
    """
```

```
    def __init__(self, startup, reinsurance_capital):
```

```
        self.startup = startup
```

```
        self.reinsurance_capital = reinsurance_capital
```

```
        self.portfolio_companies = []
```

```
class HedgeFund(object):
```

```
    """An Equity/Debt/Wrapper fund that invests in offshore portfolio companies
```

```

"""
def __init__(self, name, securitization_platform):
    self.name = name
    self.securitization_platform = securitization_platform
    self.investors = []
    self.insurance_contracts = []

def add_portfolio_company(self, portfolio_company):
    self.portfolio_companies.append(portfolio_company)

def issue_debt_or_equity(self, amount, type):
    if num_IOUs < securitized_assets_value:
        if type == 'debt':
            self.debt += amount
        elif type == 'equity':
            self.equity += amount
            total_interest_to_investor = amount * securitized_assets_value
            for investor in self.investors:
                investor.issue_interest_payment(total_interest_to_investor /
len(self.investors))
        else:
            # Redeems securitized assets

```

"""

"""

Created on Sun Mar 15 01:16:57 2020

@author: npitz

Add support for additional currencies, cryptocurrencies, stocks, and tokenized commodities, especially infrastructures, such as real estate, public transportation, renewable energy sources

inflationary_adjustment_coefficient = 1.0

price_deflation_rate = -0.0

no_inflation_rate = 0.0

producer_goods_to_produce_in_range = list(range(10, 100, 10))

service_products_to_provide_in_range = list(range(100, 1000, 100))

supplier_products_to_deliver_in_range = list(range(10, 100, 10))

class BusinessStructure(object):

```

"""Generic abstract general business structure class"""
def __init__(self, name, country, num_employees, management_style,
internal_decision_making_style, conversion_rate,
on_demand_gig_economy_mode, operational_hours_per_day,
operational_days_per_week, operational_months_per_year, profit_margin,
revenue_tax_rate, income_tax_rate, sales_tax_rate, property_tax_rate,
land_lease_rate, gig_economy_participation_rate, legal_jurisdiction,
business_structure_type, external_decision_making_style):
    self.name = name
    self.country = country
    self.num_employees = num_employees
    self.management_style = management_style
    self.internal_decision_making_style = internal_decision_making_style
    self.conversion_rate = conversion_rate
    self.internal_conversion_rate = conversion_rate
    self.external_conversion_rate = conversion_rate
    self.on_demand_gig_economy_mode = on_demand_gig_economy_mode
    self.operational_hours_per_day = operational_hours_per_day
    self.operational_days_per_week = operational_days_per_week
    self.operational_months_per_year = operational_months_per_year
    self.profit_margin = profit_margin
    self.revenue_tax_rate = revenue_tax_rate
    self.income_tax_rate = income_tax_rate
    self.sales_tax_rate = sales_tax_rate
    self.property_tax_rate = property_tax_rate
    self.land_lease_rate = land_lease_rate
    self.gig_economy_participation_rate = gig_economy_participation_rate
    self.legal_jurisdiction = legal_jurisdiction
    self.business_structure_type = business_structure_type
    self.external_decision_making_style = external_decision_making_style

    def set_external_decision_making_security(self,
internal_decision_making_security, external_decision_making_security):
        self.internal_decision_making_security =
internal_decision_making_security
        self.external_decision_making_security =
external_decision_making_security
        self.internal_decision_making_style = internal_decision_making_security
        self.external_decision_making_style = external_decision_making_security

class OptimalExchangeMethod(object):
    """A method chosen from the whole inter-universal exchange gamut"""
    def __init__(self, national_currency, cryptocurrency, stock_shares, collateral,
real_estate_property, public_utility, fiat_credit_cards, commodity_token,

```

```

external_currency, remittance):
    self.national_currency = national_currency
    self.cryptocurrency = cryptocurrency
    self.stock_shares = stock_shares
    self.collateral = collateral
    self.real_estate_property = real_estate_property
    self.public_utility = public_utility
    self.fiat_credit_cards = fiat_credit_cards
    self.commodity_token = commodity_token
    self.external_currency = external_currency
    self.remittance = remittance

```

```

class ExchangePortal(object):
    """A deep learning intelligent system that makes recommendations for optimal
    exchange methods"""
    def __init__(self):
        self.exchange_type = None

    def set_exchange_type(self, exchange_method):
        self.exchange_method = exchange_type
        return self.exchange_method

    def get_exchange_type(self):
        self.optimize_exchange_type(exchange_type)
        return self.exchange_method

    def optimize_exchange_type(self, exchange_method):
        for exchange_type in range(self.exchange_method):
            ad_hoc_essential_privacy_protection_ideal =
            optimal_algorithm(self.exchange_method)

```

```

class PersonalVirtualAssistant(object):
    """The most intelligent self-improving and auto-correcting virtual assistant ever
    built, automatically adapted over time to you. Uses intelligences far beyond human
    capabilities"""
    def __init__(self, name, remarkability, intelligence, capabilities,
    internal_decision_making_style, external_decision_making_style):
        self.name = name
        self.remarkability = remarkability
        self.intelligence = intelligence
        self.capabilities = capabilities
        self.internal_decision_making_style = internal_decision_making_style
        self.external_decision_making_style = external_decision_making_style
        self.interaction_alert = None

```

```

self.privacy_protection_option = None
self.security_default_option = None
self.encryption_default_option = None
self.desires_requested = []
self.desires_reported = []
self.reports_available = []
self.reactions_available = []
self.desire_privacy_protected = []

def set_interaction_alert(self, interlocutor_interest_level,
initiator_interest_level):
    self.interaction_alert = interlocutor_interest_level, initiator_interest_level

def get_interaction_alert(self, interlocutor_interest_level,
initiator_interest_level):
    self.set_interaction_alert(interlocutor_interest_level, initiator_interest_level)
    return interaction_alert

def issue_privacy_protection_options(self, privacy_protection_options):
    self.privacy_protection_options = privacy_protection_options

def get_privacy_protection_options(self, privacy_protection_options):
    self.issue_privacy_protection_options(privacy_protection_options)
    return privacy_protection_options

def set_security_default_option(self, security_default_option):
    self.security_default_option = security_default_option

def get_security_default_option(self, security_default_option):
    self.set_security_default_option(security_default_option)
    return security_default_option

def set_encryption_default_option(self, encryption_default_option):
    self.encryption_default_option = encryption_default_option

def get_encryption_default_option(self, encryption_default_option):
    self.set_encryption_default_option(encryption_default_option)
    return encryption_default_option

def set_desires_reported(self, desires_reported):
    self.desires_reported = desires_reported

def get_desires_reported(self, desires_reported):
    self.set_desires_reported(desires_reported)

```

```

    return desires_reported

def get_reports_available(self, reports_available):
    self.set_reports_available(reports_available)
    return reports_available

def get_reactions_available(self, reactions_available):
    self.set_reactions_available(reactions_available)
    return reactions_available

def set_desire_privacy_protected(self, desire_privacy_protected):
    self.desire_privacy_protected = desire_privacy_protected

def get_desire_privacy_protected(self, desire_privacy_protected):
    self.set_desire_privacy_protected(desire_privacy_protected)
    return desire_privacy_protected

class SecuritizationPlatform(object):
    """The securitization software platform that converts insurance contracts into
    tokenized assets and IOUs, supported by reinsurers and regulatory frameworks"""
    def __init__(self, name, country, external_decision_making_style):
        self.name = name
        self.country = country
        self.insurance_platforms = []
        self.reinsurance_platforms = []
        self.securitizations = []
        self.IOU = 0
        self.securitized_assets_value = 0
        self.securitization_fees = 0
        self.external_decision_making_style = external_decision_making_style

    def add_insurance_platform(self, insurance_platform):
        self.insurance_platforms.append(insurance_platform)

    def add_reinsurance_platform(self, reinsurance_platform):
        self.reinsurance_platforms.append(reinsurance_platform)

    def add_securitization(self, securitization):
        self.securitizations.append(securitization)

    def generate_securitized_assets_value(self, securitized_assets_value):
        self.securitized_assets_value += securitized_assets_value

```

```

def generate_securitization_fees(self, rate):
    self.securitization_fees += rate

def issue_IOU(self, amount):
    if securitized_assets_value < amount:
        print("You can't issue that many IOUs, since securitized assets
have been fully swapped away!")
    else:
        self.IOUs = amount

class Investor(object):
    """An investor that provides capital to the startup"""
    def __init__(self, name, capital_available):
        self.name = name
        self.capital_available = capital_available
        self.holds_IOU = 0

    def invest(self, amount, startup):
        if amount > self.capital_available:
            print("You don't have enough capital to invest this amount!")
        else:
            self.capital_available -= amount
            self.holds_IOU += amount
            startup.IOUs += amount

    def redeem_IOU(self, amount, startup):
        if amount > self.holds_IOU:
            print("You don't have enough IOUs to redeem this amount!")
        else:
            self.holds_IOU -= amount
            startup.IOUs -= amount
            self.capital_available += amount

    def init_IOU(self):
        self.holds_IOU = 0

    def issue_interest_payment(self, interest_amount):
        self.capital_available += interest_amount

def main():
    startup = Startup("Docker Container Builder", 1000000, 0.02)
    offshore_portfolio = OffshoreSegregatedCellPortfolioCompany(startup, 2000000)
    hedge_fund = HedgeFund("Equity/Debt/Wrapper Fund",

```

```

SecuritizationPlatform("Securitization Platform", "USA", "External"))
hedge_fund.add_portfolio_company(offshore_portfolio)

# Add investors
alice = Investor("Alice", 100000)
bob = Investor("Bob", 250000)
charlie = Investor("Charlie", 500000)

# Investors invest in startup
alice.invest(10000, startup)
bob.invest(20000, startup)
charlie.invest(50000, startup)

# Display investor's capital
print("Alice's capital:", alice.capital)
print("Bob's capital:", bob.capital)
print("Charlie's capital:", charlie.capital)

def main():
# Example usage
securitization_platform = SecuritizationPlatform("Securitization Platform 1",
"United States", "centralized")
startup = Startup("Example Startup", 1000000, 0.01)
investor1 = Investor("Investor 1", 500000)
investor2 = Investor("Investor 2", 600000)

# Investors invest
investor1.invest(400000, startup)
investor2.invest(500000, startup)

# Startup raises capital
startup.raise_capital([investor1, investor2])

# Investors redeem IOUs
investor1.redeem_IOU(100000, startup)
investor2.redeem_IOU(200000, startup)

print("Investor 1 capital available:", investor1.capital_available)
print("Investor 2 capital available:", investor2.capital_available)
print("Startup capital:", startup.capital)

if name == "main":
main()

```

It looks like you are trying to add support for additional currencies, cryptocurrencies, stocks, and tokenized commodities, especially infrastructures, such as real estate, public transportation, renewable energy sources. However, it seems that the code you provided is incomplete and contains syntax errors.

To add support for additional currencies, cryptocurrencies, stocks, and tokenized commodities, you can start by defining classes for each of them similar to the existing OptimalExchangeMethod class. You can then add methods to the ExchangePortal class to handle the exchange of these new assets. You may also need to modify the BusinessStructure class to include these new assets in the conversion rate calculation.

Here's an example of how you can modify the OptimalExchangeMethod class to include additional assets:

```
python
class OptimalExchangeMethod(object):
    """A method chosen from the whole inter-universal exchange gamut"""
    def __init__(self, national_currency, cryptocurrency, stock_shares, collateral,
real_estate_property, public_utility, fiat_credit_cards, commodity_token,
external_currency, remittance, renewable_energy):
        self.national_currency = national_currency
        self.cryptocurrency = cryptocurrency
        self.stock_shares = stock_shares
        self.collateral = collateral
        self.real_estate_property = real_estate_property
        self.public_utility = public_utility
        self.fiat_credit_cards = fiat_credit_cards
        self.commodity_token = commodity_token
        self.external_currency = external_currency
        self.remittance = remittance
        self.renewable_energy = renewable_energy
```

In this example, we have added a new asset called renewable_energy to the OptimalExchangeMethod class.

Similarly, you can define classes for other assets such as RenewableEnergy and RealEstateProperty and modify the BusinessStructure class to include them in the conversion rate calculation.

Once you have defined the necessary classes, you can modify the ExchangePortal class to handle the exchange of these new assets. For example, you can add methods to the ExchangePortal class to recommend optimal exchange methods for these new assets.

I hope this helps! Let me know if you have any further questions.

Unfortunately, as an AI language model, I do not have access to the exact formatting and requirements of brancher.ai's web page interface. However, I can provide a concatenated version of the pseudocode blueprint up to this point in a standard format:

1. BusinessStructure class:
 - Generic abstract general business structure class
 - Properties: name, country, num_employees, management_style, internal_decision_making_style, conversion_rate, on_demand_gig_economy_mode, operational_hours_per_day, operational_days_per_week, operational_months_per_year, profit_margin, revenue_tax_rate, income_tax_rate, sales_tax_rate, property_tax_rate, land_lease_rate, gig_economy_participation_rate, legal_jurisdiction, business_structure_type, external_decision_making_style
 - Method: set_external_decision_making_security
2. OptimalExchangeMethod class:
 - A method chosen from the whole inter-universal exchange gamut
 - Properties: national_currency, cryptocurrency, stock_shares, collateral, real_estate_property, public_utility, fiat_credit_cards, commodity_token, external_currency, remittance
3. ExchangePortal class:
 - A deep learning intelligent system that makes recommendations for optimal exchange methods
 - Properties: exchange_type
 - Methods: set_exchange_type, get_exchange_type, optimize_exchange_type
4. PersonalVirtualAssistant class:
 - The most intelligent self-improving and auto-correcting virtual assistant ever built, automatically adapted over time to you
 - Properties: name, remarkability, intelligence, capabilities, internal_decision_making_style, external_decision_making_style, interaction_alert, privacy_protection_option, security_default_option, encryption_default_option, desires_requested, desires_reported, reports_available, reactions_available, desire_privacy_protected
 - Methods: set_interaction_alert, get_interaction_alert, issue_privacy_protection_options, get_privacy_protection_options, set_security_default_option, get_security_default_option, set_encryption_default_option, get_encryption_default_option, set_desires_reported, get_desires_reported, get_reports_available, get_reactions_available, set_desire_privacy_protected, get_desire_privacy_protected
5. SecuritizationPlatform class:

- The securitization software platform that converts insurance contracts into tokenized assets and IOUs, supported by reinsurers and regulatory frameworks
 - Properties: name, country, external_decision_making_style, insurance_platforms, reinsurance_platforms, securitizations, IOU, securitized_assets_value, securitization_fees
 - Methods: add_insurance_platform, add_reinsurance_platform, add_securitization, generate_securitized_assets_value, generate_securitization_fees, issue_IOU
6. Other properties:
- inflationary_adjustment_coefficient
 - price_deflation_rate
 - no_inflation_rate
 - producer_goods_to_produce_in_range
 - service_products_to_provide_in_range
 - supplier_products_to_deliver_in_range
7. Other methods:
- optimal_algorithm (to be defined)

Sure, here's a summary of the total startup pseudocode blueprint in a process flow format:

1. Define economic variables and business parameters
 - Inflationary adjustment coefficient
 - Price deflation rate
 - No inflation rate
 - Producer goods to produce in range
 - Service products to provide in range
 - Supplier products to deliver in range
2. Define classes and methods for the startup:
 - BusinessStructure class with parameters such as name, country, number of employees, profit margin, etc.
 - OptimalExchangeMethod class with parameters such as national currency, cryptocurrency, stock shares, etc.
 - ExchangePortal class with a method for optimizing exchange types
 - PersonalVirtualAssistant class with parameters such as name, intelligence, privacy protection options, etc.
 - SecuritizationPlatform class with parameters such as name, country, and external decision-making style
3. Define methods for the classes:

- BusinessStructure class: set_external_decision_making_security method
- ExchangePortal class: set_exchange_type and get_exchange_type methods, optimize_exchange_type method
- PersonalVirtualAssistant class: set_interaction_alert, issue_privacy_protection_options, set_security_default_option, set_encryption_default_option, set_desires_reported, get_reports_available, get_reactions_available, and set_desire_privacy_protected methods
- SecuritizationPlatform class: add_insurance_platform, add_reinsurance_platform, add_securitization, generate_securitized_assets_value, generate_securitization_fees, and issue_IOU methods

4. Use these classes and methods to build the startup process:

- Define the overall structure of the startup
- Use the BusinessStructure class to create a business structure for the startup
- Use the OptimalExchangeMethod class to optimize the exchange of currencies, stocks, etc. for the startup
- Use the ExchangePortal class to make recommendations for optimal exchange methods
- Use the PersonalVirtualAssistant class to provide intelligent assistance and privacy protection for the startup
- Use the SecuritizationPlatform class to securitize insurance contracts and create tokenized assets and IOUs
- Use the methods of these classes to implement various business processes for the startup

Overall, this process flow demonstrates the various steps involved in building and running a startup, from defining economic variables to using classes and methods to create a business structure, optimize exchange methods, provide intelligent assistance, and securitize insurance contracts.