**Software Development from A to Z Notes:**

**Section 1 – Introduction:**

**Introduction:**

* **Top Advantages of Software Developer Jobs:**
  + #1: Fantastic job outlook, high salaries.
  + #2: Various perks.
  + #3: You will never get bored. You will always be learning new things.
  + #4: Build software products used by millions of people around the world.

**Join the Official Student Group:**

* Links and stuff. There’s also a LinkedIn page.

**Section 2 – PART 1: Software Development: Myths vs. Reality:**

**The Biggest Misconceptions About Becoming a Developer:**

* **Software Developer:** Person responsible for designing, creating, testing, and maintaining software.
* **Myth #1:** You need to be a genius.
  + **Qualities of a good programmer:** 1) Willingness to learn new things, and 2) Ability to think outside the box.
* **Myth #2:** You need to major in Computer Science. It’s nice to have, but not required.
  + **How to Learn Software Development Skills:** Coding boot camps, online courses, books.
* **Myth #3:**  It’s too late to start.
  + Software development values experience and creativity.
* **Myth #4:** You need to be a math genius.
  + Master the core algorithms and data structures.
* **Myth #5:** You need to be a jack/jill of all trades.
  + It’s better to be great at a few things. Focus on the most relevant skills and technologies.
  + No software developer knows every programming language, framework, or tool.
* **Myth #6:** You are done learning.
  + Keeping your knowledge up-to-date is imperative.
  + Learn new programming languages, explore new frameworks, solve programming challenges and quizzes.
* **Myth #7:** Your day-to-day job will be boring.
* **Myth #8:** Your job will always be exciting.
  + Some days, you’ll encounter tasks that are repetitive and boring.
* **Myth #9:** Developers work in offices.
  + Companies are embracing remote work.
* **Myth #10:** Software developers work on the world’s most ambitious projects.
  + As a software developer, you can work on projects of all sizes.

**Common Myths About Software Development:**

* **Myth #1:** Software development means writing code.
  + Meetings, communication, quality assurance, documentation
* **Myth #2:** Customers know what they want.
  + **Tips:** Create draft plans and mockups, have frequent reviews/checkpoints with customer, be honest about necessary changes to deadline and budget.
* **Myth #3:** Software can be estimated accurately.
* **Myth #4:** The mythical man-month.

**Activities Involved in Software Development:**

* Collect & analyze requirements.
* Validate ideas, feasibility studies.
* Design & prototype.
* Write code & tests.
* Write documents/guides.
* Release, support, maintenance.

**Section 3 – PART 2: Software Development Methodologies:**

**Software Development Methodologies:**

* Collection of steps required to build the software.
* Ad-hoc development leads to problems.
* Different methodologies exist.
* Waterfall & Agile.
* **Software Development Methodology:** Describes how to organize the activities involved in the software development process.

**The Waterfall Model:**

* Linear, step-by-step.
* The development process flows in cascades. Each development phase requires that the previous phase be complete.
* **1)** Collect & analyze requirements
  + Clarify with stakeholders
  + Document thoroughly
  + Probably the most important phase
* **2)** Architecture definition:
  + Components/packages Fault-tolerance
  + Key types Robustness
  + Interactions Extensibility
  + Security 3rd part usage
  + Performance
* **3)** Implementation
* **4)** Verification/Testing
  + Functional
  + Performance
  + Security
  + Usability
* **5)** Maintenance
  + Fix defects
  + Small enhancements (avoid making substantial changes during this phase)
* **Perfect Candidates for Waterfall:**
  + Life-control systems, Medical systems, Military systems.
* Use the Waterfall method if the requirements are clear and won’t change frequently.

**The Waterfall Model – Pros and Cons:**

* **Benefits of the Waterfall Model:**
  + Well-defined specification & design.
  + Early problem identification.
  + Knowledge preservation.
* **Drawbacks of the Waterfall Model:**
  + Inflexible.
  + Requirements must be well-defined early on.
  + Time-consuming architecture definition phase.
* **When to Use the Waterfall Model:**
  + Fixed scope and requirements.
  + Mature/legacy projects.
  + Technology is reliable.
  + Requires specific skills.

**Agile Software Development:**

* **Agile Manifesto (2001):** Aimed to end the proliferation of software development methodologies.
* **Agile Values:** 
  + 1) Individuals and interactions over process and tools.
    - We still need processes and tools, but they shouldn’t prevent us from implementing the required features or changes.
  + 2) Working software over comprehensive documentation.
    - But don’t use “agility” as an excuse for lack of documentation!
    - Document when it provides value.
  + 3) Customer collaboration over contract negotiation.
    - However, contracts can’t be avoided completely.
    - Focus on partnership.
  + 4) Responding to change over following a plan.
    - Don’t create a detailed plan in advance!
* Sprints and Reviews
* These frequent checkpoints are useful because they ensure the project will evolve in the right direction.
* Agile is a way of thinking.
* **Agile Frameworks:**
  + Scrum
  + Kanban

**Agile Model – Pros and Cons:**

* **Agile Development – Pros:**
  + Quick results.
  + Adaptiveness.
  + Customer satisfaction.
  + Less waste.
* **Agile Development – Cons:**
  + Inaccurate estimations.
  + Collaboration is time-consuming.
  + Requires soft skills.
  + Severe delays, inability to ship.
  + Issues caused by lack of documentation.
* **When to Use Agile:**
  + Vague requirements.
  + “Moving target” scope.
  + Need to involve the client.
  + Technology is unknown.

**Why Waterfall isn’t Dead and Agile isn’t the only Answer:**

* **Waterfall** is best suited for projects with clear requirements and fixed scope.
* Choose **Agile** if requirements are unstable and may change frequently.

**Section 4 – Agile Frameworks: SCRUM & Kanban:**

**Agile Frameworks – Scrum:**

* **What’s Scrum?**
  + Comes from Rugby.
  + Based on collaboration.
  + Simple framework for complex projects.
* **Scrum:** A way to organize teamwork around agile principles.
* **How’s work organized?** 
  + Short activity bursts called “sprints”.
  + Two – three week long activity.
  + Time for planning, communication, and reviews.
* **Scrum Teams:**
  + Teams of five to nine people.
    - Less than five: not enough skills.
    - More than nine: communication overhead.
  + Roles: Product Owner, Scrum Master, Dev Team
  + **Product Owner:** 
    - Represents the customer.
      * Communicates a lot with the client.
      * Documents the client’s needs.
    - Owns the product backlog.
      * Prioritizes backlog items, from user’s perspective.
      * Creates acceptance criteria.
    - Acts as bridge between the customer and the team.
      * Informs the team of changes, deviations, and new features.
  + **Scrum Master:**
    - Facilitator.
      * Helps in removing any blockers.
    - Guardian.
      * Protects the team from unimportant, disturbing events.
    - Coach.
      * Guides the team.
    - Scrum expert & advisor.
      * Helps team in applying scrum and agile principles.
  + **Team Members:**
    - Have complete authority & responsibility.
    - Provide and own estimates.
    - Define how to do the work.
    - Complete user stories.
    - Self-organize.
* **Scrum Team** can include the following—for example—some wearing multiple hats:
  + Product owner Developer
  + Designer Tester, developer
  + Scrum master, developer DevOps, tester

**The Scrum Sprint Cycle – Overview:**

* **Scrum Sprint:**
  + **Sprint:**
    - 1 – 3 week long development cycles.
    - Sprints follow each other without interruption.
      * Planning 🡪 Development 🡪 Review 🡪 Retrospective.

**Sprint Planning:**

* **Sprint Goals Planning:**
  + Sprint goals presented by Product Owner.
    - Prioritized list of user stories.
  + Collaboration
    - PO, team members, and Scrum master.
  + Outcome
    - Good understanding of the sprint goals.
* **Sprint Estimations:**
  + Decompose user stories.
    - Backlog items
      * Tasks
  + Estimate the backlog items.
  + Decide what fits in the sprint.
* **Time Boxed Sprint Planning:**
  + Sprint Goals: MAX 60 min.
  + Sprint Estimations: MAX 60 min.

**The Daily Scrum/Standup:**

* Short meeting
  + Max 15 minutes long.
* Answer three questions:
  + What did I work on yesterday?
  + What am I going to work on today?
  + Any issues, blockers?

**The Sprint Review and the Retrospective Meeting:**

* **Sprint Review/Demo:**
  + Team demos sprint achievements.
  + Stakeholders can inspect the product.
    - Check how it evolves.
    - Provide direct feedback.
  + Conversation between the team and stakeholders.
  + PO collects findings and creates backlogs.
* Time Box: 1 hour / sprint week.
* **Sprint Retrospective:**
  + Goal: Improve team performance.
  + Answer three questions:
    - What went well?
    - What could’ve gone better?
    - What could be improved in the next sprint?
  + Identify one improvement that can be implemented right away.
  + Real Life Example:
    - The Problem:
      * Build server running slow.
      * Affects the entire team.
    - The Root Cause:
      * Time consuming unit tests ran after each code change.
    - The Solution:
      * Lengthy unit tests will run at night.
      * Execution results get emailed to team members.

**The Kanban Agile System:**

* Introduced by Toyota.
* Goal: Improve car production efficiency.
* **High-Level Car Production Phases:**
  + Pressing 🡪 Welding 🡪 Painting 🡪 Assembly 🡪 Inspection
  + Based on supermarkets; stores only stock more items that customers are actually buying. With cars, they produce parts only when there isn’t a bottleneck.
  + Avoids overproduction, waste, and excessive inventory.
  + Needs coordination between processes.
* **Software Development Process:**
  + Feature Request 🡪 Analysis 🡪 Implementation 🡪 Testing 🡪 Release
  + If a sprint can handle three requests at a time, but the customer regularly makes five requests, the throughput will still be limited to three and a bottleneck will occur. This can lead to developers cutting corners in order to catch up. More bugs will be introduced, costing more time and money.
  + Kanban Board introduced to provide a good visualization of the Kanban process for software development.

**Kanban Boards:**

* **Kanban Board:** At the top of each board is a number that shows the maximum number of cards allowed on each board. This prevents overproduction and reduces bottlenecks on the goal.

Calendar

Description automatically generated

**Section 5 – PART 3: Programming Foundations for Absolute Beginners:**

**Programming Foundations – Overview:**

* Basic Programming Concepts
* Introducing Python
* Setting up the Environment
* Basic Syntax
* Control Flow
* Functions
* Loops
* Containers
* Error Handling
* File Input and Output
* Introduction to OOP
* Introduction to Algorithms

**What’s Coding?**

* **Programming:** Instruct a computer to execute specific tasks.
* Programs consist of sequences of commands.

**Programming Languages:**

* Examples: Python, Swift, C++, Java, C#
* **Programming Language:**
  + A collection of keywords and rules for instructing a computer.
* The CPU executes machine language instructions. These are hardware-specific.
* When we execute our high-level code, it goes through a series of transformations to make it machine-readable.

**Section 6 – Introducing Python:**

*Note: I already have Python and VSCode loaded to my machine from lots and lots of previous classes and projects, so this section is mainly just going to be review for me*.

**Setting Up Python on Your Computer:**

* Check if you have Python 3 on your computer.
* Windows: Launch CMD.
  + Type **python --version**
  + I have 3.7.3 currently.
  + Updated to 3.10.5.
* Mac or Linux: Terminal.
  + Type **python3 --version** or **python --version**

**The Python Command Line:**

* Windows: Launch CMD.
  + Type **python**.
* Mac or Linux: Terminal.
  + Type **python 3**; this starts the interactive Python shell, or “Read-Evaluate-Print-Loop” (REPL).
* **Read-Evaluate-Print-Loop:**
  + Simple, interactive computer programming environment.
  + Good for testing out short snippets of code, but not as great at editing and testing more complex code.

**Installing and Configuring Visual Studio Code:**

*Note: I already have this, but I’m going to try and configure it the same way he has*.

* Auto-complete.
* Syntax highlighting.
* **Code Editor:** A code-centric text editor that provides text highlighting, and can save and open-source files. However, it can’t execute code files.
  + Example: ATOM.
* **IDE:** More powerful. They let you execute and test the code you wrote.
* Download **VSCode**, launch it, and go to the **Extensions** panel.

**Section 7 – Basic Syntax:**

*Note: All .py programs in the following lectures and sections will be stored in location:* C:\Users\travi\SoftwareDev\_Course\_Python

**Woo-hoo! Your First Python Program!** *“first-program.py”*

* Did a standard “Hello world”.

**Asking for User Input:** *“user-input.py”*

* I added some flourishes for funsies, a la Samuel L. Jackson.

name = input("Type your name, muthafucka: ")

message = f"Hello, {name}, you muthafucka!"

print(message)

**Using Comments in Code:**

* Went over # and “”” comments. Old stuff.
* **Write Comments:**
  + To describe functionality.
  + To clarify parts of your code.
* **Comment Writing Rules:**
  + Write complete sentences.
  + Keep your comments brief.
  + Update comments along with code changes.

**Declaring Variables:** *“variables.py”*

* **Variable:**
  + Dedicated memory location.
  + Has a name.
  + Can store a value.
* **Naming Conventions:**
  + A set of rules for naming variables and other entities in source code.
* **Rules for Naming Variables:**
  + Use lower case.
  + Separate multiple words with underscores (preferred per style guide), or camel-case.
  + First character needs to be a letter or an underscore.
* **Creating Variables in Python:**
  + Provide short, descriptive names.
  + Variable names should start with a letter.
  + Use the underscore\_separated style for multi-word variable names.

**Working with Numbers:** *“numbers.py”*

* Just showed off some number values like integers and floats.
* Showed that using a comma between numbers to denote a thousand-place actually converts the number into a tuple (i.e. “42,000,000” becomes “(42, 0, 0)”).

**Using Basic Arithmetic Operations:** *“basic-math.py”*

* +, -, \*, /
* Order of operations.

**Working with Strings:** *“strings.py”*

* Concatenation of strings (plus whitespace).

**part1 = “Stay hungry.”**

**part2 = “Stay foolish.”**

**quote = part1 + “ ” + part2**

**print(quote)**

* Also, the *formatted string literal*, or **f-string**. Introduced in Python 3.6.
  + **quote = f”{part1} {part2}”**
* He then showed that these **f-strings** are good for easily formatting a string literal using variables representing strings, integers, floats, etc.
* Very nice feature. I haven’t played around with this one much in the past, but it’ll sure make things easier in the future.
* **Cool formatting trick:**
  + To display ***pi*** with only two decimals, you only need to change **{pi}** to **{pi.2f}** (or {pi.0f} if you want zero decimals).
* Also went over the following functions:
  + **len()**
  + **.upper()**
  + **.lower()**
  + **.swapcase()**
  + **.count()**

**The Boolean Data Type:** *“boolean.py”*

* **True** and **False** (**1** and **0**).
* We will go more in-depth with Boolean use in the following sections, for control flow and logical operators.

**Section 8 – Control Flow:**

**Introducing Conditional Code:** *“conditional-code.py”*

* **if-else** conditions.

balance = 5000

money = 100

if balance > money:

    balance -= money

    print(f"Current balance is {balance}")

    print("Transaction complete. Don't forget your card.")

else:

    print("Insufficient funds")

    print(f"Current balance is {balance}")

**Code Blocks:** *“conditional-code.py”*

* Indentations and such.

**Comparison Operators:** *“conditional-code.py”*

* **Comparison Operators:**
  + >, >=, <, <=, ==, !=
* Also went over the **modulo operator**, %.

**Going Deeper: Else-If:** *“conditional-code.py”*

* Used the example of a traffic light.
* Pointed out that using **elif** statements saves on computing time and resources.

traffic\_light = "flashing"

if traffic\_light == "green":

    print("Go!")

elif traffic\_light == "yellow":

    print("Slow down and prepare to stop.")

elif traffic\_light == "red":

    print("Stop!")

elif traffic\_light == "blinking" or "flashing":

    print("Proceed with caution.")

else:

    print("Invalid state.")

**Nested Conditionals:**

* Advice: Avoid deeply nested code.
* “Arrow Anti-Pattern”.
* With Boolean operators, we should be able to avoid deeply nested code.

**Using Logical Operators:** *“conditional-code.py”*

* **Logical AND Operator:**
  + True and True = True
  + True and False = False
  + False and True = False
  + False and False = False
* **Logical OR Operator:**
  + True or True = True
  + True or False = True
  + False or True = True
  + False or False = False
* **Logical NOT Operator:**
  + Negates the value of a Boolean expression.
  + not True = False
  + not False = True

**Section 9 – Functions:**

**Why Do We Need Functions?** *“functions.py”*

* **Function:**
  + A named section of reusable code that performs a specific task.
* **DRY:** Don’t Repeat Yourself.

**Writing Functions:** *“functions.py”*

* Example:
  + **def function\_name(parameter1, parameter2, … parameter\_n):**
    - # function body
    - **return value**
* **Function Naming Conventions:**
  + Function names are lowercase.
  + User underscores to separate words.
  + Names should be descriptive.
  + Start function names with a verb.
    - Ex.: “authenticate”, “calculate”, “fetch”
  + Add a noun for clarity:
    - authenticate\_user
    - calculate\_area
    - fetch\_sales\_data

**Calling a Function:** *“functions.py”*

* Just showed how to call a function and how to set its value to a variable.

**Defining Function Parameter and Return Type:** *“functions.py”*

* **Function Parameter Annotation in Python:**
  + def **function\_name**(parameter1: expression, parameter2: expression):
  + def **calculate\_square\_area**(side: int):
* **Function Return Type Annotation in Python:**
  + def **function\_name**(…) -> return\_type:
  + def **calculate\_square\_area**(side: int) -> **int**:
* **Function Default Parameter Values in Python:**
  + def **function\_name**(parameter: expression = **default\_value**):
  + def **calculate\_square\_area**(side: int = **1**) -> int:

**Understanding the Scope of Variables:** *“functions.py”*

* In this, we moved the **area** variable inside of the function instead of setting an **area** variable to be = calculate\_square\_area() outside of the function. Since we reference **area** in our f-string, we get an error, because the f-string OUTSIDE of the function can’t see the **area** variable that’s not INSIDE the function.
* A variable such as this only has a **local scope**. It isn’t available outside of the function.
* We then added “area = 0” as a *global variable* up top.
  + “area” is now defined in the beginning, but the value we’re expecting is still stuck inside the function.
  + The **local** **variable** within the function gets updated, but the **global variable** initiated at the top doesn’t. The f-string print statement still prints out the unchanged global variable of “0”.
  + Adding “**global area**” to the first line within our function fixes the result.
* As a rule of thumb, you should avoid relying on global variables in your functions. Global variables represent unprotected data (high coupling issues).

**Demo: Implementing an Area Calculator:** *“area\_calculator.py”*

* Started off by creating three functions to use later in the program:

def calculate\_square\_area(side: float):

    return side \*\* 2

def calculate\_rectangle\_area(length: float, width: float):

    return length \* width

def calculate\_circle\_area(radius: float):

    pi = 3.14

    return pi \* radius \*\* 2

**Testing Our Application:** *“area\_calculator.py”*

* Added the following lower in the program:

print("""

---------------

Area Calculator

---------------

Select a shape:

""")

selection = input("""\t'S' - Square

\t'R' - Rectangle

\t'C' - Circle

""")

area = 0

if selection == 'S':

    side = input("Enter the side: ")

    area = calculate\_square\_area(float(side))

elif selection == 'R':

    length = input("Enter the length: ")

    width = input("Enter the width: ")

    area = calculate\_rectangle\_area(float(length), float(width))

elif selection == 'C':

    radius = input("Enter the radius: ")

    area = calculate\_circle\_area(float(radius))

else:

    print("Invalid selection.  Choose 'S', 'R', or 'C'.")

print(f"The area is {area}")

*Note: The “\t” adds a tab/whitespace before ‘S’, ‘R’, and ‘C’ lines*.

**Enhancing the Area Calculator App:** *“area\_calculator.py”*

* We can enhance this program further by encapsulating all the *conditional logic* into a dedicated function.

def calculate\_area(selection):

    area = 0

    if selection == 'S':

        side = input("Enter the side: ")

        area = calculate\_square\_area(float(side))

    elif selection == 'R':

        length = input("Enter the length: ")

        width = input("Enter the width: ")

        area = calculate\_rectangle\_area(float(length), float(width))

    elif selection == 'C':

        radius = input("Enter the radius: ")

        area = calculate\_circle\_area(float(radius))

    else:

        print("Invalid selection.  Choose 'S', 'R', or 'C'.")

    return area

* We can also create a helper function to handle all the textual stuff.

def get\_shape\_name(tag):

    shape = "Unknown"

    if tag == 'S':

        shape = "square"

    elif tag == 'R':

        shape = "rectangle"

    elif tag == 'C':

        shape = "circle"

    return shape

area = calculate\_area(selection)

print(f"The area of the {get\_shape\_name(selection)} is {area}")

**Section Challenge: Calculate Rhombus Area:** *“area\_calculator.py”*

* Use: **A = (p \* q) / 2** where **p** and **q** are the diagonals of the rhombus.
* **Hints:**
* 1) Create a new function.
  + def **calculate\_rhombus\_area**(p: float, q: float):
  + Implement the code using the formula above.
* 2) Add option for rhombus selection.
* 3) Add conditional code to
  + **calculate\_area**(selection)
  + **get\_shape\_name**(selection)

**Section Challenge Attempt:** *“area\_calculator.py”*

* Added **calculate\_rhombus\_area**(p: float, q: float) up near the top with the other calculating functions:

def calculate\_rhombus\_area(p: float, q: float):

    return (p \* q) / 2

* Modified the “Selection” text prompt:

selection = input("""\t'S' - Square

\t'R' - Rectangle

\t'C' - Circle

**\t'Rh' – Rhombus 🡨 🡨 🡨**

""")

* Added an **elif** block to the **calculate\_area** function, calling the earlier **calculate\_rhombus\_area** function:

    elif selection == 'Rh':

        p = input("Enter 'p' value: ")

        q = input("Enter 'q' value: ")

        area = calculate\_rhombus\_area(float(p), float(q))

* Added ‘**Rh**’ option to **get\_shape\_name** to account for Rhombus selection:

    elif tag == 'Rh':

        shape = 'rhombus'

*Everything worked out great with these changes.*

**Solution: Calculate Rhombus Area:** *“area\_calculator.py”*

* More or less the same as what I came up with, except he used ‘**H**’ instead of ‘**Rh**’.

**Section 10 – Loops:**

**What’s Iteration in Programming:** *“login.py”*

* We implemented a simple login function. However, if login fails, it simply exits out of the program.
* To fix this, we’ll need to add looped functionality to keep prompting the user until correct login credentials are input.

**Introducing the “while” Statement:** *“login.py”*

* **while loop:**
  + Tests a conditional expression and runs the body of the loop while the condition remains true.

def login(username: str, password: str) -> bool:

    is\_authenticated = False

    if username == "admin" and password == "1234":

        is\_authenticated = True

    return is\_authenticated

user = input("Username: ")

passw = input("Password: ")

**while login(user, passw) == False: 🡨 🡨 🡨**

**print("Login failed, re-enter your credentials.")**

**user = input("Username: ")**

**passw = input("Password: ")**

print("Login successful.")

**Avoiding Infinite Loops:** *“login.py”*

* To prevent an infinite loop, we need to apply some sort of **exit condition**. In this case, we’re going to lock the user out after too many failed attempts.
* The end result is this:

def login(username: str, password: str) -> bool:

    is\_authenticated = False

    if username == "admin" and password == "1234":

        is\_authenticated = True

    return is\_authenticated

user = input("Username: ")

passw = input("Password: ")

attempt = 1

max\_attempts = 5

is\_authenticated = False

while login(user, passw) == False:

    attempt += 1

    if attempt > max\_attempts: break

    # break terminates the loop

    print("Login failed, re-enter your credentials.")

    user = input("Username: ")

    passw = input("Password: ")

else:

    is\_authenticated = True

    print("Login successful.")

if not is\_authenticated:

    print("Your account has been temporarily locked.")

**Writing “for-in” Loops:** *“for-loops.py”*

* **for** var **in** sequence:
  + # Code block that runs
  + # for each element in the sequence.
* **range() function:**
  + **range(start, stop, step)** – The start and the step are optional.
* Difference between the **while loop** and the **for loop**:
  + **while** expression**:**
    - # Runs while expression is True
  + **for** var **in** sequence:
    - # Runs for each element in the sequence

**Deeper into Loop Control Statements:** *“for-loops.py”*

* The **break** command can be used in **for** **loops** as well:

for index in range(5):

    #print("Called from within a for-in loop")

    if index == 2: **break 🡨 🡨 🡨**

    print(f"{index} - iteration count {index +1}")

* Once the index reaches 2 (the third iteration, [0, 1, 2]), the loop is broken, and the iterations stop.
* The **continue** command allows the code block to continue after skipping an iteration:

for index in range(5):

    #print("Called from within a for-in loop")

    if index % 2 == 0: **continue 🡨 🡨 🡨**

    print(f"{index} - iteration count {index +1}")

* Running **index % 2 == 0** checks indices for whether they are even integers.
* For each index that is even (**0**, **2**, **4**), the code **skips**.
* For each index that is odd (**1**, **3**), the code **prints**. Note: Index 5 is out of range.

**Rewriting the Login App Using the for loop:** *“login-using-for-loop.py”*

* Started by saving the *“login.py”* program as *“login-using-for-loop.py”*, then deleted everything below the “**is\_authenticated = False**” line. We’ll be reusing all the lines above that.
  + We also deleted “**attempt = 1**” and “**max\_attempts = 5**” variables since we won’t be using them in a *for loop*.

def login(username: str, password: str) -> bool:

    is\_authenticated = False

    if username == "admin" and password == "1234":

        is\_authenticated = True

    return is\_authenticated

user = input("Username: ")

passw = input("Password: ")

is\_authenticated = False

for attempt in range(4):

    if login(user, passw) == True:

        is\_authenticated = True

        break

    else:

        print("Login failed, re-enter your credentials.")

        user = input("Username: ")

        passw = input("Password: ")

print("Login successful." if is\_authenticated else "Your account has been temporarily locked.")

* Notice that this **for loop** version **is shorter** than when we used our **while loop** version.
* **General Tip on Choosing for Loop or while Loop:**
  + **while loop:**
    - Use a **while loop** as long as you’re checking whether a condition is **True**.
  + **for loop:**
    - To **repeat** code for a **fixed number of times**.

**Section 11 – Containers:**

**Storing Multiple Values:**

* In previous lectures, we’ve used variables to hold a single piece of data:
  + **temperature = 68**
  + **text = “Done”**
  + **is\_authenticated = False**
* However, there are situations where we need to represent **collections of items**.
* Python has **six different** built-in sequence types. We’re going to start by talking about the most common ones, the **list** and the **tuple**.

**Creating Lists:** *“lists.py”*

* **List:**
  + Stores multiple elements in a specific order.
* Simply using **list()** creates an empty list. To use it, we assign it to a variable:
  + **primes = list()**
* We can add to the end of the list with the **.append()** method:
  + **primes.append(2)** for example. We added primes 2, 3, 5, 7, 11, 13.
* We could’ve also simply added the elements at the initialization of the list:
  + **primes = [2, 3, 5, 7, 11, 13]**
* We can also create lists of **strings** (names = [“Michael”, “Dwight”, “Pam”]) or of **Booleans** (values = [True, False, False, True]).
* We can even mix datatypes within the same list:

bag = [1, 2, 3]

bag.append("Pam")

bag.append(True)

bag.append(4)

print(bag)

* This prints out **[1, 2, 3, ‘Pam’, True, 4]**

**Accessing List Items:** *“lists.py”*

* About the indices of a list, and ones that are out of range:

list = ["Michael", "Dwight", "Pam"]

name = list[2]

# 0 <= valid\_index < len(list)

def is\_valid\_index(index: int, list: list) -> bool:

    result = False

    if 0 <= index and index < len(list):

        result = True

    return result

index = 2

print(f"Index {index} is valid" if is\_valid\_index(index, list) else f"Index {index} is out of range.")

**Modifying the Contents of a List:** *“modifying-lists.py”*

* We started by initializing a list, **primes = [2, 3, 5, 7, 11]**
* Using **primes[1] = 17** replaces the **3** with 17.
  + Trying to go outside of index range causes an error.
* Using **primes.append(13)** appends the number 13 at the end of the list.
* Using **primes.insert(1, 3)** inserts the number 3 at index location 1. All other numbers move to the right.
* The **.pop()** method returns the item from the given index and deletes it from the list.
* Using **n = primes.pop(2)** returns **17**, deletes it from the list, and moves all the other numbers to the left.
  + Calling the **.pop()** method without an argument will do this to the last item in a list.
  + Calling this method on an empty list or with an index that is out-of-range will trigger an error.
* We can also remove an item without returning it by using the **.remove()** method. Unlike .pop(), .remove() expects a value, and this value *must* be in the list, or you will get a value error.
* **Common List Methods:**
  + **.append(value) .insert(index, value)**
  + **.pop(index) .remove(value)**
* VSCode version on next page:

**From Previous Page:**

primes = [2, 3, 5, 7, 11]

print(primes) 🡨 <1>

primes[1] = 17

print(primes) 🡨 <2>

primes.append(13)

print(primes) 🡨 <3>

characters = []

characters.append("a")

print(f"The length of the characters list is: {len(characters)}") 🡨 <4>

primes.insert(1, 3)

print(primes) 🡨 <5>

n = primes.pop(2)

print(f"Element {n} removed.  The primes list became {primes}") 🡨 <6>

primes.remove(5)

print(primes) 🡨 <7>

**Output:**

<1>>> [2, 3, 5, 7, 11]

<2>>> [2, 17, 5, 7, 11]

<3>>> [2, 17, 5, 7, 11, 13]

<4>>> The length of the characters list is: 1

<5>>> [2, 3, 17, 5, 7, 11, 13]

<6>>> Element 17 removed. The primes list became [2, 3, 5, 7, 11, 13]

<7>>> [2, 3, 7, 11, 13]

**Working with Tuples:** *“tuples.py”*

* **Tuple**
  + A sequence of immutable items.
  + Similar to a list, but you can’t change it.
* As a rule of thumb, use a **tuple** if you have a sequence of values that don’t need to change.
* The tuple is slightly more memory efficient and time efficient compared to the list.
* Declare a list if you must modify or update its contents.

**Storing Key-Value Pairs: the Dictionary:** *“dictionaries.py”*

* **Dictionary:**
  + Provides fast item lookup based on unique keys.
  + Stores key-value pairs.
  + A key identifies a specific item.
  + Each key must be unique.
* The **order** of keys and values is **undefined** in a dictionary.
* The short-hand syntax to create an empty dictionary looks something like this:
  + **ssn\_name\_pairs = dict()**
  + **ssn\_name\_pairs = {}**
* We can add key-value pairs to a dictionary like this:
  + **dictionary[key] = value**
* Keys-value pairs in a dictionary aren’t stored in order. To retrieve the **value** associated with a **key** from a dictionary, use:
  + **value = dictionary[key]**
  + Trying to do this with a key that doesn’t exist in the dictionary triggers a **key error**.

#ssn\_name\_pairs = {}

#ssn\_name\_pairs["123-456-789"] = "John Appleseed"

#ssn\_name\_pairs["000-000-002"] = "Dwight Schrute"

#ssn\_name\_pairs["000-000-005"] = "Pam Beesly"

ssn\_name\_pairs = {"123-456-789": "John Appleseed",

                  "000-000-002": "Dwight Schrute",

                  "000-000-003": "Pam Beesly"}

print(ssn\_name\_pairs)

print(ssn\_name\_pairs["000-000-003"])

#name = ssn\_name\_pairs["999-999-999"]

**Modifying the Dictionary:** *“dictionaries.py”*

* Dictionaries can be modified regardless of how they were initialized.
* We can **add** key-value pairs or **replace** the value linked to a particular key.
  + **dictionary[key] = value**
* If the **key** *already exists* in the dictionary, then the value will be updated.
* Using a **new key** simply adds it to the dictionary.
* We can also delete items from the dictionary:
  + **del dictionary[key]**
* We can also use the **in** keyword to check if a particular key exists in the dictionary.
  + **key = 000-000-007**
  + **if key in ssn\_name\_pairs:**
    - **del ssn\_name\_pairs[key]**
  + **else:** etc.

ssn\_name\_pairs = {"123-456-789": "John Appleseed",

                  "000-000-002": "Dwight Schrute",

                  "000-000-003": "Pam Beesly"}

#print(ssn\_name\_pairs)

#print(ssn\_name\_pairs["000-000-003"])

#name = ssn\_name\_pairs["999-999-999"]

ssn\_name\_pairs["000-000-003"] = "Angela Martin"

ssn\_name\_pairs["000-000-006"] = "Andy Bernard"

#print(ssn\_name\_pairs)

#del ssn\_name\_pairs["000-000-002"]

#print(ssn\_name\_pairs)

key = "000-000-007"

if key in ssn\_name\_pairs:

    del ssn\_name\_pairs[key]

else:

    print(f"Invalid key {key}")

**Iterating Through Containers:** *“looping-through-containers.py”*

* One of the most common tasks you will do with containers is to iterate through its elements:
  + **for item in list:**
    - **# do something with the element**

primes = [2, 3, 5, 7, 11]

#for number in primes:

#    print(number)

#index = 0

#while index < len(primes):

#    print(primes[index])

#    index += 1

even\_numbers = []

odd\_numbers = []

for number in primes:

    if number % 2 == 0:

        #print(f"{number} is even")

        even\_numbers.append(number)

    else:

        #print(f"{number} is odd")

        odd\_numbers.append(number)

print("Even number(s): ")

for i in even\_numbers:

    print(i)

print("Odd number(s): ")

for j in odd\_numbers:

    print(j)

**Iterating Through Containers, Part 2:** *“looping-through-containers.py”*

* Recall, a dictionary doesn’t store its key-value pairs in a defined order. Therefore, iterating through a dictionary doesn’t make sense.
* However, we can retrieve all keys or values of a dictionary using its **.keys()** or **.values()** methods.

ssn\_name\_pairs = {"123-456-789": "John Appleseed",

                  "000-000-002": "Dwight Schrute",

                  "000-000-003": "Pam Beesly"}

keys = ssn\_name\_pairs.keys()

values = ssn\_name\_pairs.values()

print("Dictionary keys: ")

for key in keys:

    print(key)

print("Dictionary values: ")

for value in values:

    print(value)

* There is also a dictionary method (**.items()**) that returns **tuples** of key-value pairs.

key\_value\_pairs = ssn\_name\_pairs.items()

print("Key-value pairs")

for key\_value in key\_value\_pairs:

    print(key\_value)

* One method:
* Alternatively:

key\_value\_pairs = ssn\_name\_pairs.items()

print("Key-value pairs")

for (key, value) in key\_value\_pairs:

    print(key, value)

**Demo: Extract Duplicates:** *“find-duplicates.py”*

* We’re going to implement a program that goes through a dictionary of **ssn\_name\_pairs** and prints out a new dictionary that takes duplicate names as its **key** and lists of SSN numbers as its **values**.

ssn\_name\_pairs = {"123-456-789": "John Appleseed",

                  "000-000-002": "Dwight Schrute",

                  "999-000-005": "Pam Beesly",

                  "888-888-888": "John Appleseed",

                  "999-000-006": "Pam Beesly",

                  "999-000-007": "Pam Beesly"}

# Want:

#{"John Appleseed": ["123-456-789", "888-888-888"],

# "Pam Beesley": ["999-000-005", "999-000-006", "999-000-007"]}

def find\_duplicate\_names(ssn\_name\_dictionary: dict) -> dict:

    result = {}

    names = list(ssn\_name\_dictionary.values())

    for (ssn, name) in ssn\_name\_dictionary.items():

        if names.count(name) > 1:

            result[name] = result.get(name, [])

            result[name].append(ssn)

    return result

duplicate\_name\_ssns = find\_duplicate\_names(ssn\_name\_pairs)

for (name, ssns) in duplicate\_name\_ssns.items():

    print(f"Found duplicate name {name} with SSNs: {ssns}")

**Section 12 – Error Handling:**

**Getting Ready for Errors:** *“errors.py”*

x = 10

for i in range(100):

    x -= 2

    print(i / x)

print("Hello!")

* After 5 iterations, x is now 0, and we get a **ZeroDivisionError**.
* Code never even makes it to printing “Hello!”.

**The Buggy Equation Solver:** *“equation-solver.py”*

def solve\_equation(a: float, b: float, c: float) -> float:

    return (float(c) - float(b)) / float(a)

print("ax + b = c linear equation solver")

a = input("Enter a: ")

b = input("Enter b: ")

c = input("Enter c: ")

x = solve\_equation(a, b, c)

print(f"x is {x}")

* Although our code has no issues of its own, if we input ”0” for **a**, we get a **ZeroDivisionError**. The case of a = 0 isn’t handled. We can’t control what the user enters.

**Handling Errors:** *“equation-solver.py”*

* Python allows us to “try-except” a piece of code to handle possible errors:
  + **try:**
    - # code that might fail
  + **except ErrorType** (*ArithmeticError, ValueError, IndexError*)**:**
    - # code to execute when the error condition occurs
* We may also encounter **try** statements with more than one **except** clause:
  + **try:**
    - # code that might fail
  + **except ErrorType1:**
    - # code to execute when the error condition occurs
  + **except ErrorType2:**
    - # code to execute when the error condition occurs
  + **except ErrorType3:**
    - # code to execute when the error condition occurs
* This strategy is called **Structured Exception Handling**, and can be useful if you expect multiple exceptions and want to handle them separately.

def solve\_equation(a: float, b: float, c: float) -> float:

**try:** 🡨 🡨 🡨

        return (float(c) - float(b)) / float(a)

**except ZeroDivisionError:** 🡨 🡨 🡨

**print("Error! 'a' can't be zero. Enter a valid value.")** 🡨 🡨

**except ValueError:** 🡨 🡨 🡨

**print("Error! Make sure you enter numeric values.")** 🡨 🡨 🡨

print("ax + b = c linear equation solver")

a = input("Enter a: ")

b = input("Enter b: ")

c = input("Enter c: ")

x = solve\_equation(a, b, c)

print(f"x is {x}")

**Raising Exceptions:** *“equation-solver.py”*

* Using **raise** can help catch errors before the code finishes running:

def solve\_equation(a: float, b: float, c: float) -> float:

    try:

        return (float(c) - float(b)) / float(a)

    except ZeroDivisionError:

        print("Error! 'a' can't be zero. Enter a valid value.")

    except ValueError:

        print("Error! Make sure you enter numeric values.")

**raise** 🡨 🡨 🡨

print("ax + b = c linear equation solver")

a = input("Enter a: ")

**if float(a) == 0:** 🡨 🡨 🡨

**raise ValueError("'a' can't be zero.")** 🡨 🡨 🡨

b = input("Enter b: ")

c = input("Enter c: ")

**try:** 🡨 🡨 🡨

    x = solve\_equation(a, b, c)

**except Exception:** 🡨 🡨 🡨

**print("Something bad happened.")** 🡨 🡨 🡨

**else:** 🡨 🡨 🡨

    print(f"x is {x}")

**Cleaning Up: the “finally” Keyword:** *“equation-solver.py”*

* Used like this:
  + **try:**
    - # code that might fail
  + **except ErrorType:**
    - # code to execute when the error condition occurs
  + **finally:**
    - # code that executes before leaving the function
* Using this, we can leave a **trace message** such as “Leaving solve\_equation()” to help us track down bugs.

def solve\_equation(a: float, b: float, c: float) -> float:

    try:

        return (float(c) - float(b)) / float(a)

    except ZeroDivisionError:

        print("Error! 'a' can't be zero. Enter a valid value.")

    except ValueError:

        print("Error! Make sure you enter numeric values.")

**finally:** 🡨 🡨 🡨

**print("Leaving solve\_equation()")** 🡨 🡨 🡨

print("ax + b = c linear equation solver")

a = input("Enter a: ")

b = input("Enter b: ")

c = input("Enter c: ")

try:

    x = solve\_equation(a, b, c)

except Exception:

    print("Something bad happened.")

else:

    print(f"x is {x}")

* We will be learning more practical applications for the “**finally**” keyword in the next “File Input and Output” section.

**Section 13 – File Input and Output:**

**Working with Files:**

* Files: Used to store data on a persistent data medium.
* Python makes it easy to work with files:
* **Reading a File:**
  + Check if the file exists.
  + Perform validations.
  + Read and process contents.
  + Close the file.

Diagram

Description automatically generated

* File I/O operations may fail for various reasons.

**Writing Text to a File:** *“file-write.py”*

* 1) Open the file.
* 2) Write data.
* 3) Close the file.
* Our **write\_file** function will take a filepath (string) and text (string) parameters, and output an integer representing the number of characters.
* There are many open-modes we can use when opening a file:
  + **r** – open for reading 🡨
  + **w** – open for writing
  + **x** – create new file
  + **a** – append new content
  + **t** – open in text mode 🡨
  + **b** – open in binary mode
  + **+** – open for update
* If we don’t choose a mode, the file opens for **reading** in **text** mode by default.
* In this case, we’re going to choose **w** because we want to **write** to a file.

def write\_file(file\_path: str, text: str) -> int:

    file = None # handler

    try:

        file = open(file\_path, **"w"**) 🡨 🡨 🡨

        return file.write(text)

    except OSError:

        print(f"Error! Couldn't open file at path {file\_path}")

    finally:

        if file != None: # checks handler

            file.close()

filename = "myfile.txt"

count = write\_file(filename, **"Hello Python!"**) 🡨 🡨 🡨

if count != None: # checks handler

    print(f"{count} characters written to {filename}")

else:

    print(f"Could not write to file{filename}")

import os

print(os.path.abspath(os.getcwd())) # prints our current working directory

* If we pass the name of a file we’re creating without a **file\_path**, it will be created in the same location as our program (our working directory).

**Reading Text from a File:** *“file-read.py”*

def read\_file(file\_path: str) -> str:

    file = None # handler

    try:

        file = open(file\_path)

    except OSError:

        print(f"Error! Couldn't open the file at path {file\_path}.")

    else:

        lines = []

        for line in file:

            lines.append(line)

        return lines

    finally:

        if file != None: # checks handler

            file.close()

text = read\_file("myfile.txt")

if text != None:

    print(text)

**Section 14 – Introduction to Object-Oriented Programming:**

**What’s Object Orientation?**

* **1950** – **Unstructured Programming:** Contiguous chunks of code consisting of sequentially ordered instructions.
  + Coding sheets.
  + Punched cards.
  + As programs grew in complexity, the drawbacks of this approach became apparent.
* **1960** – **Structured Programming:** Breaks down code into logical steps and relies on functions.
  + Even with the improved readability and quality, developers began to face new challenges as the programs became more sophisticated.
* **1980** – **Object-Oriented Programming:** Organize the program into discrete objects.
  + **Object:**
    - Represents a distinct entity.
    - Operates on its own data.
    - Has a specific role.

**Understanding Objects:**

* **Object-Oriented Programming:**
  + A programming paradigm organized around self-contained objects.
* **Object:**
  + Represents a “thing”.
  + Has its own data and logic.
  + Interacts with other objects.
* Objects can be described using their **properties**.
* **Properties:**
  + Color
  + Size
  + Shape
* Ex.: **Engine:**
  + Petrol
  + Diesel
  + Electric
  + Hybrid
* Ex.: **Finish:**
  + Solid color
  + Metallic
  + Matte
* This approach to describing real-world objects by their properties also works in programming languages.
* **Object:**
  + State
  + Behavior

Diagram

Description automatically generated**Introducing the Class:** *“class.py”*

* Identify the objects.
* We need classes to create objects.
* **Class:** The blueprint of an object.
* We can use the example of Pokémon to create a class, a general blueprint that can be used to describe multiple Pokémon instances.

class Pokemon:

    def \_\_init\_\_(self, name: str, max\_armor: int, max\_hit: int):

        self.name = name

        self.armor = max\_armor

        self.hit\_points = max\_hit

    def attack(self):

        print(f"{self.name} attacks")

    def defend(self):

        print(f"{self.name} defends itself")

pikachu = Pokemon("Pikachu", 100, 1000)

pikachu.attack()

pikachu.attack()

pikachu.defend()

snorlax = Pokemon("Snorlax", 75, 900)

snorlax.attack()

charmander = Pokemon("Charmander", 1200, 600)

charmander.defend()

**Abstraction:**

* **Abstraction:** A way of describing complex problems in simple terms by ignoring some details.
* **Define a Class:**
  + Focus on essential qualities.
  + Skip details that are irrelevant.
* In our Pokemon example, we focus on essential qualities (name, armor, hit\_points, attack(), defend()). We didn’t need nonessential details like “age”, “weight”, or “height”.

**Encapsulation and Data Hiding:** *“class.py”*

* We encapsulate something to protect its data and to keep its parts together.
* **Encapsulation:** Packing together variables and functions in a class.
* **Data Hiding:**
  + Avoid exposing internal details of a class.
  + Reveal only the essential features.
  + Protect objects from unwanted changes.
* **Private Visibility:** Restricts access to variables and methods to the class that defines them.
* Unlike other programming languages, there’s **no private keyword** in Python.
* All class instance variables and methods **can be accessed** by callers.
* Instead of relying on a special keyword, Python relies on **naming conventions** to “hide” instance variables and methods.
  + If a variable or method isn’t meant to be seen on the outside, we precede its name with an underscore (i.e. “self.**\_is\_charging** = False”, “ def **\_change\_attack**(self):”)

class Pokemon:

    def \_\_init\_\_(self, name: str, max\_armor: int, max\_hit: int):

        self.name = name

        self.armor = max\_armor

        self.hit\_points = max\_hit

**self.\_is\_charging = False** 🡨 🡨 🡨

    def attack(self):

        print(f"{self.name} attacks")

    def defend(self):

        print(f"{self.name} defends itself")

**def \_change\_attack(self):** 🡨 🡨 🡨

**print(f"{self.name} is changing attack type")** 🡨 🡨 🡨

pikachu = Pokemon("Pikachu", 100, 1000)

pikachu.attack()

pikachu.attack()

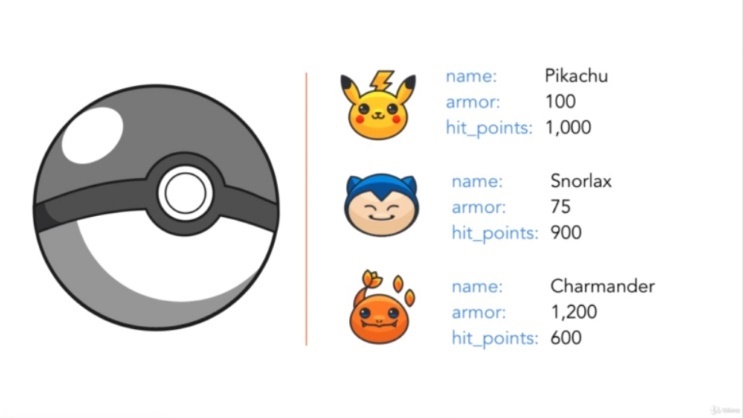
pikachu.defend()

**pikachu.\_change\_attack()** 🡨 🡨 🡨

* Although we can access these private methods outside of the class, Python developers know that this is a bad idea. The private method’s behavior could change, or they could be deleted someday, for example.
* **Data Hiding:**
  + Prevents accidental modifications.
  + Reduces external dependencies.
* Data hiding isn’t about selfishly keeping stuff for ourselves, it’s about protecting our classes from unwanted external dependencies. If we restrict access to the data or behavior, we don’t have to worry about the ripple effect from making changes.
* **As a rule of thumb:** Expose only what’s required for the normal usage of the object.

**Inheritance:** *“class.py”*

* **Inheritance:** Code reuse. Reusing an existing class’s implementation in new classes.
* Graphical user interface

  Description automatically generatedOur basic Pokemon attributes: New Pokemon types with new attributes:
* The basic Pokemon class has properties “name”, “armor”, “hit\_points”, and methods “attack()” and “defend()”.
* The “electric”, “water” and “flying” type Pokemon will also have all of these general properties/methods, along with some special ones of their own.
  + Electric type also has “**wild\_charge()**”
  + Water type also has “**aqua\_tail()**”
  + Flying type also has “**dragon\_ascent()**”
* We could add all these new methods to the **Pokemon Class**, but then we’d end up giving ALL Pokemon these methods, and the class will have too many responsibilities.
* **A class should have one, well-defined responsibility**.
* Instead, we’ll keep these classes separate:
  + **class ElectricPokemon**
  + **class WaterPokemon**
  + **class FlyingPokemon**
* Inheritance allows us to skip re-writing the code that’s shared by all these related classes:
  + **Class ChildClass(ParentClass):**
* Code shown on following page:
* Code from previous page:

class Pokemon:

    def \_\_init\_\_(self, name: str, max\_armor: int, max\_hit: int):

        self.name = name

        self.armor = max\_armor

        self.hit\_points = max\_hit

        self.\_is\_charging = False

    def attack(self):

        print(f"{self.name} attacks")

    def defend(self):

        print(f"{self.name} defends itself")

    def \_change\_attack(self):

        print(f"{self.name} is changing attack type")

**class ElectricPokemon(Pokemon):** 🡨 🡨 🡨

**def wild\_charge(self):**

**print(f"{self.name} used Wild Charge attack!")**

**class WaterPokemon(Pokemon):** 🡨 🡨 🡨

**def aqua\_tail(self):**

**print(f"{self.name} used Aqua Tail attack!")**

**class FlyingPokemon(Pokemon):** 🡨 🡨 🡨

**def dragon\_ascent(self):**

**print(f"{self.name} used Dragon Ascent attack!")**

* By claiming “Pokemon” as the parent class for each of these, we only needed to write each subclass’s unique attack. Standard properties and methods from the parent class will be automatically inherited by the subclasses.

**Method Overriding:** *“class.py”*

* The example here: The **Water type** Pokemon inherits its “**attack()**” method from the Pokemon parent class. However, what if we want its attacks to do **more damage** against Electric type Pokemon?
* **Method Overriding:**
  + Changing the behavior of inherited methods in a child class.
* Method overriding is straightforward: We re-implement the method with the same name and parameters as the one defined in the Parent Class, and provide our own behavior.

class Pokemon:

    def \_\_init\_\_(self, name: str, max\_armor: int, max\_hit: int):

        self.name = name

        self.armor = max\_armor

        self.hit\_points = max\_hit

        self.\_is\_charging = False

    def attack(self):

        print(f"{self.name} attacks")

    def defend(self):

        print(f"{self.name} defends itself")

    def \_change\_attack(self):

        print(f"{self.name} is changing attack type")

class ElectricPokemon(Pokemon):

    def wild\_charge(self):

        print(f"{self.name} used Wild Charge attack!")

class WaterPokemon(Pokemon):

**def attack(self):** 🡨 🡨 🡨

**print(f"{self.name} special attack!!!!")** 🡨 🡨 🡨

    def aqua\_tail(self):

        print(f"{self.name} used Aqua Tail attack!")

* Re-implementing the **attack()** method with new behavior overrides this method for the Water type only.
* Now, what if we want to add a new “Speed” instance variable in the Water Pokemon class.
* We need to create a new “**\_\_init\_\_()**” with all of the same parameters as in the Parent class, and we need to add a “**speed: int**” parameter at the end.
* Within the \_\_init\_\_, we then need to re-initialize the Parent class parameters on their own, and then the new parameter below that (“self.speed = speed”).
* When creating a new WaterPokemon, we need to pass it an extra parameter on the end for speed:

class Pokemon:

    def \_\_init\_\_(self, name: str, max\_armor: int, max\_hit: int):

        self.name = name

        self.armor = max\_armor

        self.hit\_points = max\_hit

        self.\_is\_charging = False

    def attack(self):

        print(f"{self.name} attacks")

    def defend(self):

        print(f"{self.name} defends itself")

    def \_change\_attack(self):

        print(f"{self.name} is changing attack type")

class ElectricPokemon(Pokemon):

    def wild\_charge(self):

        print(f"{self.name} used Wild Charge attack!")

class WaterPokemon(Pokemon):

**def \_\_init\_\_(self, name: str, max\_armor: int, max\_hit: int, speed: int)** 🡨 🡨

**Pokemon.\_\_init\_\_(self, name, max\_armor, max\_hit)** 🡨 🡨 🡨

**self.speed = speed** 🡨 🡨 🡨

    def attack(self):

        print(f"{self.name} special attack!!!!")

    def aqua\_tail(self):

        print(f"{self.name} used Aqua Tail attack!")

class FlyingPokemon(Pokemon):

    def dragon\_ascent(self):

        print(f"{self.name} used Dragon Ascent attack!")

**vaporeon = WaterPokemon("Vaporeon", 99, 1000, *20*)** 🡨 🡨 🡨

**Polymorphism:** *“class.py”*

* **Polymorphism:** The condition of occurring in different forms.
* In programming, we can use this to iterate through all of our Pokemon, regardless of type, and have them all attack. **It lets us work with instances of classes that share a common superclass**.

^^^

class ElectricPokemon(Pokemon):

    def wild\_charge(self):

        print(f"{self.name} used Wild Charge attack!")

class WaterPokemon(Pokemon):

    def \_\_init\_\_(self, name: str, max\_armor: int, max\_hit: int, speed: int):

        Pokemon.\_\_init\_\_(self, name, max\_armor, max\_hit)

        self.speed = speed

    def attack(self):

        print(f"{self.name} special attack!!!!")

    def aqua\_tail(self):

        print(f"{self.name} used Aqua Tail attack!")

class FlyingPokemon(Pokemon):

    def dragon\_ascent(self):

        print(f"{self.name} used Dragon Ascent attack!")

pikachu = Pokemon("Pikachu", 100, 1000) 🡨 🡨 🡨

vaporeon = WaterPokemon("Vaporeon", 99, 1000, 20) 🡨 🡨 🡨

raichu = ElectricPokemon("Raichu", 98, 800) 🡨 🡨 🡨

togekiss = FlyingPokemon("Togekiss", 75, 1200) 🡨 🡨 🡨

pokemons = (pikachu, vaporeon, raichu, togekiss) 🡨 🡨 🡨

for pokemon in pokemons: 🡨 🡨 🡨

    pokemon.attack() 🡨 🡨 🡨

* **Output:**

**Pikachu attacks**

**Vaporeon special attack!!!!**

**Raichu attacks**

**Togekiss attacks**

**Section 15 – Introduction to Algorithms:**

**The Importance of Algorithms:**

* **Algorithm:**
  + A sequence of steps describing what the computer should do.
* The study of algorithms is essential. They can be found everywhere.
* **Why Should You Study Algorithms?**
  + Process large amounts of data efficiently.
  + Solve complex problems.
  + Succeed in job interviews.
* **Algorithmic Thinking:**
  + The ability to find the most efficient technique to solve a problem.

**The Problem with Naïve Implementations:** *“sum-n.py”*

* We’re going to solve the same problem using two different approaches:
* **Implement a function** that calculates the sum of the first N natural numbers:
  + 1 + 2 + 3 + 4 + … + N
* To measure our time efficiency, we’ll be using Python’s “time” module.
* **Python Module:** A file that contains Python source code.

import time

def sum(n: int) -> int:

    result = 0

    for i in range(n+ 1):

        result += i

    return result

ranges = (10, 1000, 10000, 100000, 1000000)

for n in ranges:

    start = time.process\_time\_ns()

    sum(n)

    end = time.process\_time\_ns()

    exec\_time = end - start

    print(f"sum({n}) execution time: {exec\_time}ns")

* ![Graphical user interface, text

  Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4SS6RXhpZgAATU0AKgAAAAgABgALAAIAAAAmAAAIYgESAAMAAAABAAEAAAExAAIAAAAmAAAIiAEyAAIAAAAUAAAIrodpAAQAAAABAAAIwuocAAcAAAgMAAAAVgAAEUYc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFdpbmRvd3MgUGhvdG8gRWRpdG9yIDEwLjAuMTAwMTEuMTYzODQAV2luZG93cyBQaG90byBFZGl0b3IgMTAuMC4xMDAxMS4xNjM4NAAyMDIyOjA4OjAyIDE0OjU3OjM5AAAGkAMAAgAAABQAABEckAQAAgAAABQAABEwkpEAAgAAAAM3OAAAkpIAAgAAAAM3OAAAoAEAAwAAAAEAAQAA6hwABwAACAwAAAkQAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAyMjowODowMiAxNDo1NzowNwAyMDIyOjA4OjAyIDE0OjU3OjA3AAAAAAYBAwADAAAAAQAGAAABGgAFAAAAAQAAEZQBGwAFAAAAAQAAEZwBKAADAAAAAQACAAACAQAEAAAAAQAAEaQCAgAEAAAAAQAAEw4AAAAAAAAAYAAAAAEAAABgAAAAAf/Y/9sAQwAIBgYHBgUIBwcHCQkICgwUDQwLCwwZEhMPFB0aHx4dGhwcICQuJyAiLCMcHCg3KSwwMTQ0NB8nOT04MjwuMzQy/9sAQwEJCQkMCwwYDQ0YMiEcITIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIy/8AAEQgAVgEAAwEhAAIRAQMRAf/EAB8AAAEFAQEBAQEBAAAAAAAAAAABAgMEBQYHCAkKC//EALUQAAIBAwMCBAMFBQQEAAABfQECAwAEEQUSITFBBhNRYQcicRQygZGhCCNCscEVUtHwJDNicoIJChYXGBkaJSYnKCkqNDU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6g4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2drh4uPk5ebn6Onq8fLz9PX29/j5+v/EAB8BAAMBAQEBAQEBAQEAAAAAAAABAgMEBQYHCAkKC//EALURAAIBAgQEAwQHBQQEAAECdwABAgMRBAUhMQYSQVEHYXETIjKBCBRCkaGxwQkjM1LwFWJy0QoWJDThJfEXGBkaJicoKSo1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoKDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uLj5OXm5+jp6vLz9PX29/j5+v/aAAwDAQACEQMRAD8A8crTttFlureKVZUHmZwpznv+fSgBY9EkklaMXEW5BluuAME9fwqM6TIskyNIgMSbz784xQBMdBnVmV5YlwxVcn7/AE6f57GsogqxB7GgC3aae92oKyKpZiqg55IGTT00xmyGmRWGcAg8/Nt/nQA6bSJIZNrSofvZIzxtGTmql1B9muHh3hypwSBQAttb/aXZd4QKpYkgngVfGgzeasRniDkE8npzigCJNImZ5VLqoiIDEg9TjH8xTI9MkdlQyKsjOUVSD2ODQBMuiTOiyJLG0bDdu5GB6kenag6JKHVRNGS+dnXDYGTzQBVksmitTOzrxIU29+Kq0Ab0Xhl5rZJkvYSrojgEEH5iwA5/3TVpvA9/56xrPAcy+WxJI28Zyfb/ABHrQBH/AMIXqG6MLLEwchSVydvzFeePYn6Vj6lps+l3IguABJgkgdsEj+lAFOigAooAKKACigAooAKeJZBjDsMDA56CgCUX10Mf6RJxnHzetH225IwZ5DkY+92oABe3IbInkzjGd1QE5OaAHpPNEjJHIyq3UA9aVrmd23NK5bGMk/jQAC4mVHQSsFflhnrUbMzsWYksepNADopZIX3xOyNjGVOKkF7dLjFxIMdPmoABe3QKkXEmVGB81NW6nRSqzOFJ3EBu/rQA43lyX3meTd67jSm9ujnNxJz1+agBkl1PKpWSV2UnJBPeoqANGKW7jTEeooilQuBKeg5A/U/nThNegYGqKBnP+uPrn+YoAPOvCGH9qLhxhh5x5HP+J/Oo7h7u5WKGS8Fx8wVE8wtg/jQGwv8AYmo/8+x/76H+NH9h6j/z7n/vof41p7KRl7en3F/sPUf+fY/99D/Gj+w9R/59j/30P8aPZSD29PuH9h6j/wA+x/76H+NH9h6j/wA+x/76H+NHspB7en3D+w9R/wCfY/8AfQ/xo/sPUf8An2P/AH0P8aPZSD29PuH9h6j/AM+5/wC+h/jWewKsVPUHBqZQcdyo1Iz+ESipLCigAooAKKACigAooAKKACigAooAKKACrFh/yEbb/rqv86a3Jl8LPSbG0S6S6LuymGEyLgZyQQMfrUt5p8Ftptjcx3JkkuFYtHsxtwccHvXffU8uytcz6KZAUUASxW08yM8ULuq/eZVyBUVJSTbS6FOEopNrRhXnk3+vk/3j/OsMR0OrCbsZXQ2FzoIsLaO6tx9oXJkk2scnJwOvTHtXMdo25l0Xz4zbRKtv+8yHjYtnDbcnPT7tZNw9rIsf2eB4nAw+X3A+4oA3pbzw7P50jRMrGDaF2HPmYPzAg+u39aqzXmjvBdeTbJFIGPkbVbOONpzng5znPtQA66m0uW3uLjyvMmlnwgU7W2kAnjsQRjp3NOsX0OeOO2ngMJLfNMzZZR82TngHHynGKAFt59GKxGeNQgaUlNjnaDjb9SMH9KjW60uKFESKFi1syu7RNkPkEd/wyKAFuLvRUjzbWiPIUUDerbR83Pfrjv0qHWXsbvWozbusUDKgciPaqcDOABnj86ALEN1oknmyXEQ8zzAsabTjYCmM49g351G8+j/aAscI8pJXkZipDMo5VBzwD0/GgB9nf6ZMkR1JNzIZPl2HA3FT2I7bh9SKfbS6BuTzo/3fk427GL7uM5OevpigCBrywKrcuTJL5CQeXt5B6M3oeBx9fars8/h4Wk0sEMZl8wbF2tk45HBPA9c9eaAIEudMe/ckQ+VczksNhCpGBkDHXk+npUv2jQ7aS3uo4I2LTFWQgkBAOW25zyTwD2WgBIr7QZFgW6t1LRJtLqjYbnnIB9OlJbXGjme2e3tgs0Z2BSD85IUAn3B3n8qcdyZfCzolYqcqSM8HBrS1D/kEaT/1zk/9GGu9nlLZmZRTJCnIjSMFUZJpN21Gld2RbS9ezSSG3f5XUrIf71UqiEEm5dWa1ajklDotgrmPO0mRI2ljjiuEmZ5Aqt84GcD8eB+NZ1+hthN2YIGSAO9dK2lxnw0sYhjF95q88biPmyc+nKjHtWEYuWx1Smo7lqIPPb2lldW5a3hhCgA7vn3ZJxnHIyPxqxAtpFOsiaYEZE+Vgin+Fc9+ud3Paq9lIn20B3l6VHGqHSA7eWFYsqgnn69cd6q3Udud32fTIFJOVOwcdMdSffNHs5B7eA+Rbf8A0idLT/SmuRNGyqq7V3A4GD6ZFRLZWzIC9rvlW4aVTtA8wckK3Prgcds0eykHt4FiKeXy1+0I0itD5b25AKbt4LN17gHtUKxvsjUswAtJLdgEXjJO0degz+lHs5B7eAyxijtbO6hfTSjXEfl7klzgckHn/a2njHSrcUFpMsk95brJIfLLB8FnwFBAOeOjc980ezkHt4FKa0BlBtrGMKZULSGMdABnAzxkg8VawybtloHxLOV3Rrlg44BPt6UezkHtoD2TTWkQDS0RFJz8q5xt+vXPrVawjS0mhD24eOGMFTgE7yct3+gz7UezkL28C0BpgjZW0iI5UAYUZHTvnr15qnc20Id5LW2QeYVRlKhRsDZPAPf5R+B9aPZyH7aBZhg0yO5llfTt6k/IhVcfw+/s3HvWKdR0tdQkkGmoE2MBnkbtuB8vTGamUWtyozUthbGfR0vpZ5lPllXPlvGGAO4YA5/u5+lTLqdvb3NjHpiJG0uPtBAzyTjb+AH5k0o7lS+FnR1p6h/yCNJ/65yf+jDXezyVszMopkjkRpHCqOTUjusaGKI5z95vX/61S9XY0j7seYhoqjMK89n/AOPiT/eNc9fodmE3Y2P/AFi/UV6Fb3dslmkUsJZgTlgB0IP/ANaopdTTEdC22pwTOFitNv3gNqjIBUj+uaW0v/s8ag280rCAjJXtnP5dq2OYoTCR5I38iTbHGu7K9QO/0q7JqNqszB7EfeU7SoGAOcfjQBE+oWzuT5GAXyQEX5hgD8KX+0ofNhIjZY4pd4UKvI44/T9aAHx6hboihbQsSMLuRfXn655qjb3IheY4+WRCvABx+dAF+HUrZI8G1MuI1Vtyjt/Kq17dRTGTZD5e8LtGwDHX/GgB1jqMdrAsckbSASbwOBj3zWmus+bhlspWDMeVXqcHn689KAKxvVlmVmsGwd2MRjJ65rGeN0ClkZQ3KkjqKAG0UAFcZL/rW+prKr0OnD9RlWbE/wDEwtv+uq/zrGO6N5fCzva09Q/5BGk/9c5P/RhrvZ5S2ZmU5EaRwqjJNDdlcSTbsiV3WNDFEcg/ef1/+tUFKK0uypvWy2QUVRAV57P/AMfEn+8a56/Q7MJuxsf+tX6ivQ7e8hSySGSBnwxJYYHBBqKPU0xHQtNqYncIloVzvxjGQCpH9adaXslvGgW0nkYQHk88ZzkcdO1bHMTyapcNCy/2fKMw8HHTIIz06c1mTNIl8txJbNsi8sFJBjIAAGfrigCcapFvLfZd3zhskD06dPx/KmNfxSE7oH2sWyFAGcjr060AON+JpYswy7Y9+EXHQ5Pp704alGiRgWnATBzjkcZ7ex/M0AUbe5MEM8TBsTKBxx3zWiuppFlTZM2NpcSAc8Ac8e360AUbu5FwZCImUs4PQDtjsKntbyNoobdrV5WjPAQgHufTPf8ASgDRi1S5Xy1+wSswBBOOSQevTr61C+oTyRwr/Z74VMn5eCOM446cfrQBnagzSNETbvFiMfeGM+9UiCOoxQAVxkv+tb/eNZVeh04fqMqxYf8AIQtv+uq/zrFbm8vhZ31aeof8gjSf+ucn/ow13s8pbMzkRpHCqOTUjuqIYozwfvN/e/8ArUnq7FR92PMQ0VRmFFABXns//HxJ/vGuev0OzCbsbH/rF+or0KC8aOySJrcugJ+bOOoORUUeppiOhbfUbi4YKbUqDuwORwVIP6ZpbO5u7dFWK0Zj5Ock9s5z/wDWrY5i1/aOpbeLH/lnkc/XP9ePaqd895OkzyWpQbVGc8Kuc/zoAghvpbSH7IYQWWTceBknjjp7VKdUnZfL+zEZYkhRzznocZzzQBGL+VVRlt3GAVLbj83BH9adDdS2yR5tWOI8gsfvDOfyoAr3U891tZoWXYM5wehqdL6ZpJbryHflNx3EgEdP5UASzancSRmNrUj5wWKjkkY7468VnJI0N0J2jJ2vna3r6UAXopLkxwKsGWLGRSzjLE8ZH41bi1G+SGFVtVICEKQ45Ix/h096AEnutQljANmVXbu4PXgj/E49qztSFxLdNJLbtG2Bnv8A/WoApMpUlWBBHUEVxcv+tb/eNZVeh04fqMqxYf8AIQtv+uq/zrFbm8vhZ31at6jSaVpKqMkxyf8AoZrvbseXFX0RRd1jQxRHIP3n9f8A61QUorS4TavZdAoqiAooAK89n/18n+8a56/Q7MJuxiHDqT2Ndva6662MSpp9xLEWO1/LOGzkEAj/ADxWVOSjub1abnaxNL4qmmI32M+1i+AIz6ENj6DP5VWg8YRWcaoluyny9uWB55yD1rT2sTH2EjQl8V3cJdWspPkjBJVCQB6/Tmo7jxRPLbyRzWM0cbAAs0ZAQH39KPaxD2EinLrf2TU5LhreTdE5zhcqMcdfxFWF8WSeT5wspjCWJ37DjoQefp/Kj2sQ9hIcviSTckhsZnZgy7dpyQCCePY96ij1qSVRIbC4dVgbnaQCnc/hmj2sQ9hIe/iS4lARbC4A2biBG3IIIz9Oapy6yumebZ3MEqs21mU9R3Hf3o9rEPYSLCeOY49wSDCsckEE/wBfaqd14ot7r7ySD5y3r1AHr7Ue1iHsJEp8XxGGCMxsfJGFOPYgd/emweLLeBkYRMdiFFHbnqTz70e1iHsJFv8A4TwbSBFjjjA6H16/pTJvHCTMcxFVJzgA/wCNHtYh7CRQuvEVrc3DzFGXd2ArmHO52I7nNRUkpbG1Km4XuNqxYf8AIQtv+uq/zrNbmkvhZ31aH9pK1jDaywbhECAyvtJBOcHj1r0LJ7nk80l8JB5tp/z6v/39/wDrUebZ/wDPq/8A39/+tVXj2M7T7h5tn/z6v/39/wDrUebZ/wDPq/8A39/+tRePYLS7h5tn/wA+r/8Af3/61L5toTgWj5P/AE1/+tRePYLS7iXoQTjy4GhG0ZVj1Pc15pP/AK+T/eP865sTuduC2ZHWxaeI7yztYrZUjaOIEKGBzyc9c9a5TvGXOuz3d0LmaGIygMMjcAQ2e2e241SuL65uo4455mkWPO3d1H40Aan/AAlN6Q+6OFnaLyd+0g7cEY688E/p6VWk1y6ljuEdUPnszFufl3Yzjn2FAD3164NpJFHmKSSYSs6H2xx6Z6n6CpLLxJdW0iGcC4RSCyufv43cH2+Y5oAjg1+4gKsIo2dXdyxLfMWxnIzgjgcVE2sSsqL5MYVYTDtGQCDznr1zzQA+5166uYTEyoqMBu255Oc56+tQanqUmp3KzyKqlUVOCTnA7k8mgClmjNABmjNABmjNABRQAVNaOsV5DIxwqupJ9s01uKSujrf7e0//AJ6n/vk0f29p/wDz1P8A3ya7PbQ7nnfV6nYP7e0//nqf++TR/b2n/wDPU/8AfJo9tDuH1ep2D+3tP/56n/vk0f29p/8Az1P/AHyaPbQ7h9Xqdg/t7T/+ep/75NKNfsFYMJjkHP3TR7aHcPq9TsPk8R2UxBeYnGf4D3Oa4uUhpnYdCxIrGtNStY6MNSlTvcZRWB1BRTsAUUWAKKLAFFFgCiiwBRRYAoosAUUWAKKLAFFFgCiiwBRRYAoosAUUWAKKLAf/2f/hMehodHRwOi8vbnMuYWRvYmUuY29tL3hhcC8xLjAvADw/eHBhY2tldCBiZWdpbj0n77u/JyBpZD0nVzVNME1wQ2VoaUh6cmVTek5UY3prYzlkJz8+DQo8eDp4bXBtZXRhIHhtbG5zOng9ImFkb2JlOm5zOm1ldGEvIj48cmRmOlJERiB4bWxuczpyZGY9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkvMDIvMjItcmRmLXN5bnRheC1ucyMiPjxyZGY6RGVzY3JpcHRpb24gcmRmOmFib3V0PSJ1dWlkOmZhZjViZGQ1LWJhM2QtMTFkYS1hZDMxLWQzM2Q3NTE4MmYxYiIgeG1sbnM6eG1wPSJodHRwOi8vbnMuYWRvYmUuY29tL3hhcC8xLjAvIj48eG1wOkNyZWF0b3JUb29sPldpbmRvd3MgUGhvdG8gRWRpdG9yIDEwLjAuMTAwMTEuMTYzODQ8L3htcDpDcmVhdG9yVG9vbD48eG1wOkNyZWF0ZURhdGU+MjAyMi0wOC0wMlQxNDo1NzowNy43NzU8L3htcDpDcmVhdGVEYXRlPjwvcmRmOkRlc2NyaXB0aW9uPjwvcmRmOlJERj48L3g6eG1wbWV0YT4NCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIDw/eHBhY2tldCBlbmQ9J3cnPz7/2wBDAAMCAgMCAgMDAwMEAwMEBQgFBQQEBQoHBwYIDAoMDAsKCwsNDhIQDQ4RDgsLEBYQERMUFRUVDA8XGBYUGBIUFRT/2wBDAQMEBAUEBQkFBQkUDQsNFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBT/wAARCAHFBUIDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwD896KKKACiiigAoo60UAFFFHHc4oAKKQ+3NLQAUUfjR9OaACij9KKACij8aKACigDPSigAopeNuc0n40AFFHXtR+OKACijnuMUu0+lACUUfhijaewzQAUUdKKACiiigAooooAKKKPwoAKKOB1OKOD0OaACijkHGKMHcBj8aACig4HTmlwPWgBKKKP1oAKKTNLz3GKACilx68UnWgAoo6dRRx64oAKKB+VL+NACUUUUAFFFFABRRRQAUUvVsA5pM/hQAUUc/wB00UAFFFFABRRx2OaKACik3AAE8Up/OgAooooAKKOPWigAoo/HFH45oAKKXHy56UhypxQAUUcdN1HTvQAUUfTmkycE4oAWijjGc8+lAGelABRRRQAUUvHrSe9ABRR+FA56DNABRQfl69aKACiiigAooooAKKKKACigDNHfA5oAKKXafSk6UAFFFHpQAUUc88UcBgCcGgAopcHtzRj14oASihfmz6etJuHrzQAtFHHrilI24560AJRRnk+nrQuW6CgAoo+oxS4LdBQAlFHPYZpP0NAC0UUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAOXJOPm/Cjy5C2CjY+leqfAjwnp/ivxCYNQi82EHJHHpXb3l18PLLWm06ayMbK2wtxQB86SKRzjHsKbtOMiNvrXt3xK+FOn6TcWV9pbB7C4YN8voa6vxHovgjwPpli19Z7pJwOgHp/8AWoA+aO38Q+tJnPG819GN4F8JePvD891oAWGeNC23HP8AKvPfhj8MT4r1yaO7Pl2lu2HY+1AHnHlv2VnpjRhSCcr/ALNfQ+qX3w68O3RsBbrOwba74rP+Inwt0fUPDbeIPDbBoNu4on8NAHhCrv6K34UpVo/v7xXvvwv8JaA3gafVdWtvNMS5LHGc1peH7P4f+Mro2EVqIrg/KDgdfyoA+cFj3fdzupNhBIbO6vWR8PbXSvigujylXtTJjHtWJ8YfDtl4c8WXFrZpshU8DsOKAOAVeoxlu1OMUi4O1ge/Fbfg7T49T8Q2dvP9x2ANe/8AjLT/AAJ4JuIba6ssuVGSAMdKAPmVwUA3bgfTFJtPVoyB616n401TwhqUCRaTZ+VMTjcMV6Bqvwd025+GEepWUI+2CLcWxyaAPmza34Uojz2b8KlFu32wxYO/O0A19E+Hfg7p0fw1l1a+hzeeTuHHINAHzl5ZY8/N7CnfZ3xkxso9a9p+Evwx03Wob/U9RXNtb5Oz8q3NN1D4e61qTaSbVbd87VkYcZoA+dkxzgHPvSda7j4peE7XwzrjRWUyy27HKla4gjBwKAEooooAKKKKAClVQ2aStbw3oE3iLWLezgGWkYD6UAZYUscLuH4UPGy9UYf7Rr6M1Tw14M+GdjFHqard3+PnQdqWHwp4Q+JGj3P9jqsN+q5Cd80AfOI+YYClvpQuGzyQR2Net/C3wDa33jK403U4fljJG1hXEfEPSbfR/Et5a23yxqxAoA5sZY8ZApyRuyEqGNeu/B3wNpniTQ9UmvIjJJCpKn34o8K6r4P06F7bVbIvcBtoPGTzQB5G0br8oVvypuzbgEsDX1Trnh/wJovhm31eaw+SZcjGK86s7fwx4u8YWFpp9mY4Xb514wRzQB41gnojfXFO8ts4fPtX0l4tt/h/4T1Y6fNZ7WBwSAPSsLx18NtFv/DC+INAP+j43MvpzQB4SwdP4Tj1psnbIO72r2vxt4C0zS/hjbapBHi6ePLNivJdFtlu9StoX4RnANAGeoO4A7hxmlZH3Y2NXunjzwroHg/UNMlktgbeRQXC9O9dH4Es/AfjTURaw2WHA54FAHzQY2zwGX8KDnGM5P0r3vxdf+AtNkurOOyxOuQDgcGvCbxlaZ2j/wBWTxmgCCiiigAooooAKKKdH96gACSOu4ZI9qTYRwVYmvoPwD4R8MwfD46zq8G91XPNTaXa/DrxUxgi8u1mbhdw6n8qAPnXB6ZJ9qcqSHgKwr0bxz8N38IeKYIcGSxnkGx16Yr1nxJ4d8FeC9Kspb2x+aRASeKAPmBsp97cKQbm6BiK9d8Ua94JutNkTTrLbcnhW4rrfDfhfwrp3w/j1rVLTzG2biePWgD532E8bGx600RnfwpavpDw3oPgb4gSS2Nhb+RdgHaTiuP8D/D+1b4iz6LfxiSGNyN3rxQB48ykHDL0pVUyL8o6V0vxC0230XxFdWsHEaMcAV1fwd8IWHibTdWmukVjEp257dKAPLlI3Ywc0/yX6ojV6h8OfhaPF2v3Zl/d2Fu53N7V2epap8PfD91/Zv2QTFTtaRcYoA+ezkDv75pua9u+JXw+0WTw1/b3h6RRFt3PHnpXiW35tp4IoAEDN0PPpSmNx95HHvXtPwn+Fun3GlNreuFUskG7Dd63LLWPh3q2oHTvsgiGdomOKAPng4yQcijluisc8cV6f8TvhvFoOtWyae6z21y3yeX713lr4E8K/DrQIbvxCFmu2XPldzQB87hGT7yMpPcik8tl6fOTX0doml+BfiOrWNnGlnekfJkd687t/h2NJ+IiaTfKTD5m32IoA80IaP7ylc0BSeEBavffjj8FYfDdmt9pUQaEDc6qOgxXFfBPwvYeKvEv2W+jzHg9enSgDzfaFP8AED6UrRSD5tshX6V7X4P+Ftn4k8bXtu2I7KBz8vsK3rjVPh9pesHS5bH7rbC7AYoA+dtvGWyKRV3NgBvwr1X4reBdO0m8hk0WVJ4Lg/Kq9s112jfDvw54J8Lx6r4j2tcSLlYT1NAHz+yvH1RgPU0zlunzGvo7Q4/h/wCPXayhhW1uGGELDjNcG/w2Hh/4jRaTdpvtZHwrDoRQB5htPXG0+lRMdrcjn0Fel/G7wtY+FfEH2eyj8uMDotcHotut1qdsjnKMwBoAqLA5BO1h3HFIyyL97cK+l/Emk+CvBNnZC+sxJJKoOVA9K8/8Za94LutJkXTLEpcsMK4xkUAeTUU6T71NoAKKKKACiiigA/4EQfSpPJk25Eb/AJVY0lol1CIzg+UGBYivsT4V6p8HvFV5pukTaOWvJSEZgByaAPjH7PIekJP1BpfLbONvzfQ1+gnxm0T4N/CPVE0+70fEjLxwK8+/Z98A+CPib451e4ksC2kxZZIzjgcUAfHwhlXA2f8AjppPLcNtYHJ6cV9oa34y+B2l6pNanRCXjcocbeorkPBWg+B/il8bLGx0qx2aXJLhozjpigD5eNvNg5ik456VHIxVc8g+jV95/EP/AIUv8O/Fkuh32kYaM7WkwMVxPxw+BfhC+8AJ4y8GFTa43sijpz9KAPkRVcoCyttP8VG044OVr708O/sr+H/Fn7Oa63aWv/E58jfuA5zXwrfabLY6jLayRtHJHIVKkdeaAK3lu3HzEDrtFIoaRSEDbelfdn7Nv7KuleI/hLf+IPEFkWmeAyQkgcVw/wACPgt4f8W2/i1r+DzWs9/k5xxjFAHyZ5LNHhdzEdTTCp5zyPWvt79lf9njw18TbHXVvoB5sRYIz446V86/Hb4P3nwo8XXNpJC32YyEIccEUAeWCNtuRGxHrTvLH4+gr6ktfhHoUn7NLeJvs6jVPJ3byOc5rxL4N+C/+E++IGn6U8TOkkoD7R2oA4eRXiUEoQPU0gXcwCqz59K+3P2u/wBmPQPA/gq11bwxEsjQRg3Hl4689a8G/Zk8Fad42+IlnpepxCW3lbBXsOKAPHWXb94bGpvH/Aq9U/aO8G2Hgv4kajpenIEt42wuBwK8sYgscduKAEooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA9w/Zhj8zxVIOhx/StXxJ8C77VPFs9211HHB5m8ksOKwf2bb6Gx1+VppFjUjqx9q5r4heMNUTxPerDeyGEyEDa3GKAPUfihr2n6fYaToNrcJcvDtRvL+proPiV8O4vGWkaNJJqKWhVBhXP19q+ZtHvJrjWraSaQuS4yXNev/HrxA0NlopsLrGxRkIe/NAHZ6b4bsfgr4Xu5ri5WeSaM7COhrnvhzdtJ4G1y+gGJmDMu3r2qTwzrFr8SfhvPpuozql7BHtVnPJNc78HPElppGpXnh/UpVS0uDs3v90f5xQB49qEjS30zyMWbceTXv8A8C5XvfBGqW05Y26RHIbp2qvrf7PK32oST2Go28lnI24Mh+7Wj4kvdN+E/geTSLO6Se/lj2OyHvQBq/DOx0+++HepQ3sywWpzuZ+ijNM8BeH/AARo+qmSDVLe4u85UKGzn8qpfC60j8T/AA1vLIXEdvPMmPmPOcis/wAOfA06XrUV5ca1EiK24jdQBBfWl7H8aIJbwbd8vyH14rjf2g5RH46usnIz/SvQPEXiix1D4tWEcEqyQ27gNLnjpVr4mfBf/hMNenv4NSgCS4YDd0oA8N+Hq58WWBHXeOT9a+ivix4V8MatqUL6tqsNrNgZVgeDj6V5V/wrKXwD4o02SS7jmUyDJDdK9N+Kvwvk8b3kVza6lDGhAON3tQB5L4w8L+GdE8l9I1KO7kLZ2oD/AFFe/wCj+JItP8K6HYXJzBcR7Pr1rwLxV8JbrwjHFcPfQ3CAgkKea6z4ia8lj4R8PS2kqmWMAkA9OtAGFN8MJv8AhaH2RUJtvN35xxjrXtOqa9Fd6DrGn2zbYrWPaMHjtVZfF2jL4JTxKWj/ALQWLJX+InGK84+G+vJqGh+IpbmVY3lBZdx+lAFD4UfFK08Kteabqab7W4JDN2HNdRL8OPCXjSZp9D1eGG7Y7lTkHP5Vzfw48JeHfGdle2txPHFqAzsdvWt3wt8Db3w1r8V6+rRJbRtuyrdR6UAeS/EPwpqfhbVGt9QlaXB2q7VyTD15r1z9ojxNa69rgS0lEgjO1seteR0ARUUUUAFFFFACjC7s8V6n+z7FHL4zt920MBkA+teWfeXJrqfh74m/4RTxHa3hOFRvmoA1PjLfXF342vzJuU7uVP4Ve+AV1Mvjq1WIsEJ+ZfWvTfGfwz074mSLrOl30Mc0oyyk1J4P8C6f8J4ZdY1K+hluo8mONT7dKANLR7aG1+MN35LAlmOV/CvA/i1z4yvyRjDnpXVeGfictr8Sjq077YZXwxz0Fdp42+DkHja8k1bSb6BxcfMdx5FAFL4AZ/4RnWm6r5Zx+leHap82uSMMZEv9a+ivsNh8IfA99Ab2Oa8mTG1D06f4V83ST/aNQMhGQ75oA97+JGf+FN6ZjjEXB/OvN/gsP+K/07P98ZzXoHxB1G2m+EelxJMrSLHyufrXnXwZmWHxxYvK6qm7JLelAHr/AMV/gvqHijxTLdJJHHE7Z5YdMU3xNJp3w9+GzaGLtLi7kG0oh75ri/jd4wvrfxdKLO9YwZwNjcdK8qudYudQmVriV5SDn5jQB9B/ENhJ8FbHngxfd/E14L4a/wCQ5adgHFe9eDLuw+JngH+wp7lILiNNqbz3qr4b/Z7XSdWS8v8AU4IraNt3XrigCl+0Zxb6UO4QfyNYn7OP/I4E4A4I/Sq3x68X2+ua/wDZbVvNgtjsDjoeKf8As9XcNn4tMksqxpg8sfagDj/iKxHiq/YFsiU8GuXbnGcg4rpfH0iTeKL5kbcrOea5r1Gc4NACUUUUAFFFFABTo/vU2nR/eoA+grVi3wJuAM7vK/rXhNjcNa6nAEGXDAhvQ19HeB9Ii8VfCVtMW6hhmZMfMeax9A+Adlpl8lzqeqweTGdxANAGx8QALjwVoF1cnNzhT79TXS/EzQPD+taPpo1fUI7T92Nu8Hnj2FeU/FL4gWusa5YabZN/oNq4UMOhx/8Arr0z4h+BB8QNG0wwahDFsjGfm570AePeMPCXhHS9PMmmapDczLyFUH/CvVvDulaTqXwfgh1O7jtrYx/eYH1+leUeJ/gndeHtPluzqEUyR/MQrc16ToPh5fFnwmt9NivYoZ1XadxoA0fhvovhPS5pDo2oxXGot91VBznHTpXNeA7G/t/jBc/2gGWRnPH4VofD34R/8IXrUepX2rw+XGd20N1rG1X4mWEHxW+3xvi2D4LL37UAeW/FZT/wml8BxhyCTXoP7Px26LrxOB+77/hXTeOvgzb+O9R/tfS72Bo5vmYE9Kd/YunfB3wTfK93HNdzJgqh6dKAJfh9M1j4D16eAYmCseOvUV4n4b8OxeK9dmhuLtLQ7iS8meefavQPgt48sY5rzSdRkVILolcsfXFamofs+yNqT3Gm6nCttK24Nu5AoAqa18Kb3R/B1xPbawlzaKmWRM4x+VeFxrtugrc7nxmvo3x9q9j4A8Bvokd2t5dTJsZlOcV83bmEjnvncKAPpLxlMNP+B9sLdsAw/Nt69a+b45nhmidXYHqOetfQ/wANta0vx14Hfw3qNysU2zagc8Gs6x/Zv8nU1ludQgSzByDntQBx3w81i617xZpsN67NGrDaX6VuftIXE0niwREsIUwAO3SoviFqGieDfEVnHom13tmG+VOhr0DxF4a0r41aPDqFpdxQXgUbo2POaAPDPhreS2fjCylifcWkH3f5V7v8QIx/wtDTHxtZmG78qo+Bfg/Y+Ab9tX1a/hMMAyI89a5m/wDHUXin4qQzRELarJ8rse1AHpGueNrceNLnw/qjZtZx5aljx0FYvgT4fzeC/iOyopNpJlkbtjFebfHPVIx46a6tpQ21gQ6H2r2r4OfErT/E2isupTJHfW8fyux5P+c0AeaaX8Rk8CfEbUXniMlvJIc47dK6TUvDvgj4hz+dZalFZ3sh3bWznP5Vg6Jpnh7xN421WDUZEVpHPlu3TtVlf2fbu11YXFtqcS2gfcrBu1AGNrHw81Dwf4o01Lydrix8weWx6EVa/aSvJvt1rbhSsCqNnPB4rc+OHiy002LSLKG4W7ntiNzqfTNXr/StL+NXhi0kjuY4dQiTkOefSgD588G3Etvr1o8Tt5m8c19MeObdW8W+HZ2I859u7H41zPhT4E2vhXUBqGq6nCYoTvCg+lZ2qfEO38TfFKySCQC0glAVieMUAYn7Sr7fFhBPzA/0rzDw3j+2bQt/fBP519KfFT4U/wDCea8L2DUrdY255PtXlWu/CeTwTeWly95DOu8ZCn3oA9j+Jnhvw5qun6VJq+ow2suwYVwfQ+1eN+OPCvhXS9LaXTdVhup1HCRg5z+Ir1j4h/D8ePbHTZra9hTYgyC3Tg15T4q+DVx4b0mS9e/il2jO0GgDy+T71Np0n3qbQAUUUUAFFFFACqvzZIB9q9N/Z4X/AIulo3O3bKCAteZLgZPfHFekfAG4itfiZo7zOEQSLlj0oA9n/b9+bxxDzyAPvfSui/YBhguNU1FbhlSEodxbpjArkv28NStNS8bWxt5Y7hQAS0Zz2ra/YNuLaTUNRtZ7uO2MyFVMjYxwKANXxV8GfhHc6/eST+LbKGbexZNr5Bz0+7XP/A3w9oOg/tGWVt4dv47+xSXiWPOCMdeQK1/FH7Geqaxrd3eReIrUxzSltvmdqx/g18P2+EHx2sLS/v4ZFEmPM3cYwaAPWvjd+yrbfEn4lTXK+I7a3uJWz5LE5H6Vh/HyXS/gP8KY/AcVwt3fTKEZkB/z2ry79oT4p3/hz47T6lpt400Ec4IEbcdq9C+PzaV8ZfhXYeLLS4jXUraNXkjLDcT0/rQB7N8J/iInhD4N+GYrth9lvIgj56ck18wfFv4Hyt8d4LayhJsr2cOgxwwPNdV4/wDElvY/s7+HViuVW5t0BKK3OQTXrvw68d+G/EXwvsvF2qXEH9r6dBlUc/MTgj+tAHoei+LLTT7G98H6bhUsrcrJ5f3fu/8A16+fv2XlH2Xx9xjHmY9/u81mfs1/ENNf8V+Kr7UbtIhJv2+Y3XgVP+zNrlhYw+OVnu44mk8zGTjd0oA6D9lvUptD8H+MLiBikkZZgw7crVfX5NP/AGpfhxdxxeWfElipARfvsQf/AK9ZXwE16xs/AXjGKe7SKR1fapYZbkV84/B74wXXwp+IUl/DuFs058xc8MM0AfSV9ot14f8A2SLmzvI3iniiMbK31rmf2Gfh7cSXWo+J/srSyW6louByeP8AGvcfj5438N+NfgD9o0ieBZLuIb4VPIJNcQfiNB+z38A7GPRZo31a4i+coRnPP+FAHWeDfDvib4heGfFul+IbC4jRw3k+cQd3TpzXzV+zP4dk8P8A7Qyae6+S0U5GwjB711Pwh/bQ8Sah42s7XU322M77XOex/GvQJ9K0rQ/2m9O1e0uYXtL2TcZFYYXINAHzb+17Jt+L2qgjcN/OPpXiOd2WHCV97fHD9lGT4keOLrWbXW7RIZ2yF39OK+YPjL8B7r4T+SJb6G7EhwDEc+tAHk/8WO9FOGdwzxTRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFACg7aSiigC1Y6lPp7boHKN6g1DNcSXDlnYsx7mo6KAHrIVYMOo71ZutSuNQ2ieRpQvQMelU6KALdnqVxp6EW8zpk5OD1qH7VJ53mByHznNRUUAbtt421iziKRXkirjGN1Zt1qVzqTbrmZpX/vMeaqUDigDVs/EF/pqgW9zJEPRTT5fFmqT533khB96zPvCmeWaALQvJ47jz0lYS9d2eTWh/wlmrH/AJfpR+NZFGecUAX7zXL6+ZWnuXkZeQSelWF8WasvH22TH1rIooA0L3Xr7UIwk9xJIB6mq9xfXFzEI5ZWdF6KT0qvRQBaOqXJthb+a3k/3M8Uy3vri1haOOZlRuoB61BRQBNZ3k+nzebbytE/+ya2JPHGsyQtE15JtI9awaKAJLi4kupGeVixbrmowNvA6UUUARUUUUAFFFFAB2A9KdvOc02igDV03xPqOkqq291IoAwBmk1HxNqOqptubl5F9zWXRQA7zGznPNbNh4x1bTYRFDeSKo96xKKANC+1i81bJup3l9mNVOQ2QSOMVFRQBdm1e6uLYW8krNEBgKTwKhtbqS2mWSJjG69CtQUUAW7y8lvpjLM5d+uTVXcR0PFG0tzSUAXdP1K401xJbytE3sa07jxxq91F5cl9IyYxtzXP54I7UUASyzNLks5Y+9SWN9NZyb4JGif1U1WC56CgqVoAkmuHuJC7sWY85NR+vvzRRQAUUUUAFFFFABSqdtJRQBo2PiC+02MJb3DxKOymppvFWpXKMkl3IVPvWRRQBL5nIO7JzmtVfF+qxgKt5IFHTDVi0UAa03ibUryMxy3Ujo3UE0238SajZx+VDdSRp6KazAcHNJQBrTeKtTmjCPdyFf8AerOa4aRgdxBFQ4zS0Abem+MtWsYNkV3Iq9hniq2pa5e6xuN1O0oPYms2lyfu0ALG5hkDxnaw6MOtbcfjjWYIdiXkgGMferCpQN1AFi51K41Bi9zI0rf7RqvnKketJRQBPa309jMskEjRspzlTW03jjWJovLa9kIxjGa56igCaa5e5kLStuJ6k1d0/Xr7ScfZrh48ehrMooA2L/xdqupR7Li7kkXGME1mxXMtu4lRyHzwahooAnuLqS9bzJ3LtnkmpbO/uLD5reVoz6qap0/cAvFAEi6hdQ3AmEzCXOd2ea218ea20fl/bpdnpmudPzdaRTQBbvL6e8k8yaVpHJzljU+m6/faS2badouc/KazqKANq68ZateRlJbyRlxjk1k29xJbzeajsH9c1HRQBtf8JbqqxjF7L8v+1Va61++1AAXFxJIB/eNZ1FAGyvizVIsKt3IFHQZ6VHeeJdRvIjFNcySx4xtY8VlZ5xS+lAAzbqSiigAooooAKKKKAD8Oans7ybT5llgkMcinIYHkVBRQBd1TVrvXLgSXk7zv/eY5qTTPEGoaFNvsLl7dvVDis9fvUN980AdL/wALM8TYx/ak/wD33VCbxdq1zfLeS3krXC/dk3cisiigCxfapc6lM01xK0sjclmPNXIfEeo2+nm0jupFtmGDGDxWXRQBp3Wv6hdWYs5LmR7cDiMngUW/iTVLaxe0iupI7ZuCitxWYzY6fSl5DAZ60AXrHXL7R8m0uZIS33tpxmn2PiPUbDzDBdSR+Z97B61mtlqKANO18SahZRyJDcuiyfeAPWs3cd27qc5PvSUUAazeJdTXTRZi8kNsowI88U3UPEuo6rbJDc3Uksa/dRjwKy6KAJLO5ezmEsXyupyDWxJ4w1fz45zeymSPhTu6Vh9FzRuyoFAHWD4neJV6arcD/gVZes+KNS1xVN5dyTYGBuOayCaM/LigQjZbGe1FFHNBQUUvNHNK6ANpo20c0c0XQrBto20c0c0XQWDbRto5o5ougsG2jbRzRzRdBYNtG2jmjmi6CwbaNtHNHNF0Fg20baOaOaLoLBto20c0c0XQWDbRto5o5ougsG2jbRzRzRdBYNtG2jmjmi6CwbaNtHNHNF0Fg20baOaOaLoLBto20c0c0XQWDbRto5o5ougsG2jbRzRzRdBYNtG2jmjmi6CwbaNtHNHNF0Fg20baOaOaLoLBto20c0c0XQWDbRto5o5ougsG2jbRzRzTANtJS80lABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABQOOlFFAEm8Ubh61HRQBJuHrRuHrUdFAEm4etG4etR0UASbh60bh61HRQBJuHrRuHrUdFAEm4etG4etR0UASbh60Fx2qOigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigBdpOcKT9KkFnOefJkxtxyhNLZTi1uUmK7ip6dq9f0346abY2ccLaEHKjBbA/xoA8f+x3P/PCT/vg0fY7n/nhJ/3wa9p/4aA0v/oXx+Q/xo/4aA0v/oXx+Q/xoA8W+x3P/PCT/vg0fY7n/nhJ/wB8Gvaf+GgNL/6F8fkP8aP+GgNL/wChfH5D/GgDxb7Hc/8APCT/AL4NH2O5/wCeEn/fBr2n/hoDS/8AoXx+Q/xo/wCGgNL/AOhfH5D/ABoA8W+x3P8Azwk/74NH2O5/54Sf98Gvaf8AhoDS/wDoXx+Q/wAaP+GgNL/6F8fkP8aAPFvsdz/zwk/74NH2O5/54Sf98Gvaf+GgNL/6F8fkP8aP+GgNL/6F8fkP8aAPFvsdz/zwk/74NH2O5/54Sf8AfBr2n/hoDS/+hfH5D/Gj/hoDS/8AoXx+Q/xoA8W+x3P/ADwk/wC+DR9juf8AnhJ/3wa9p/4aA0v/AKF8fkP8aP8AhoDS/wDoXx+Q/wAaAPFvsdz/AM8JP++DR9juP+eEn/fBr2n/AIaA0v8A6F8fkP8AGj/hoDS/+hfH5D/GgDxf7HcH7sEg/wCAGka1mg5eJwB1JUjFe2L+0BpYUAeHx+Q/xrnfG3xYsfFGlS2kGk/ZZG6Px/jQB5js6rncvXNHrTrf97PGp4ViAa9r034R2FxYpMzDLKCeK6KdF1Ohy1sRCh8Z4gck0vPtXug+Dem/3hR/wpvTvUV0/U5djk/tLDroeF8+lHPp+hr3X/hTmneo/Kj/AIU7p396j6nLsL+0qHY8M20ba9y/4U3p/wDfo/4U3p/9+j6nLsL+0qHY8N20ba9y/wCFN6f/AH6UfBvT/wC/mj6nLsH9pUOx4Zt9qNte6/8ACndPH8Qpf+FO6f8A3h/n8KPqcuwf2lQ7HhO2jbXu3/CndP8A7w/z+FH/AAp3T/7w/wA/hR9Tl2D+0qHY8J20ba92/wCFO6f/AHh/n8KP+FO6f/eH+fwo+py7B/aVDseE7aNte7f8Kd0/+8P8/hR/wp3T/wC8P8/hR9Tl2D+0qHY8J20ba92/4U7p/wDeH+fwo/4U7p/94f5/Cj6nLsH9pUOx4Tto217t/wAKd0/+8P8AP4Uf8Kd0/wDvD/P4UfU5dg/tKh2PCdtG2vdv+FO6f/eH+fwo/wCFO6f/AHh/n8KPqcuwf2lQ7HhO2jbXu3/CndP/ALw/z+FH/CndP/vD/P4UfU5dg/tKh2PCdtG2vdv+FO6f/eH+fwo/4U7p/wDeH+fwo+py7B/aVDseE7aNte7f8Kd0/wDvD/P4Uf8ACndP/vD/AD+FH1OXYP7SodjwnbRtr3b/AIU7p/8AeH+fwo/4U7p/94f5/Cj6nLsH9pUOx4Tto217t/wp3T/7w/z+FH/CndP/ALw/z+FH1OXYP7SodjwnbRtr3b/hTun/AN4f5/Cj/hTun/3h/n8KPqcuwf2lQ7HhO2jbXu3/AAp3T/7w/wA/hR/wp3T/AO8P8/hR9Tl2D+0qHY8J20ba92/4U7p/94f5/Cj/AIU7p/8AeH+fwo+py7B/aVDseE7aNte7f8Kd0/8AvD/P4Uf8Kd0/+8P8/hR9Tl2D+0qHY8J20ba92/4U7p/94f5/Cj/hTun/AN4f5/Cj6nLsH9pUOx4Tto217t/wp3T/AO8P8/hR/wAKd0/+8P8AP4UfU5dg/tKh2PCdtG2vdv8AhTun/wB4fl/9aj/hTun/AN4fl/8AWp/Updh/2lQ7HhSpk0rIexr3Nvg7p+3IYflXlfjbRY/D+rSW0RyoPBrCrh3TV7HRRxdOs7ROcbIbFFOb0/Gm1yHcFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFACt1o/ixQ1L/ABGpewn0H2oIuIuP4v619S6Jn+x4f90V8t2xJni/3v619S6L/wAgeHH90V7WC2Pns1vdFsjpRtNO9KK9mx8w4u43aaMGnZ9qQc9OaLBysTdRup1FOxPLIbuoyetOoosKzHbqM0bRRtFFg1DNLSbRRRYNRaTNFG0UWDUM0tJtFGPeiwai0maMe9G0UWDUM0tJtFFFg1FpM0UbRRYNQzS0m0UY96LBqLSZox70bRRYNQzS0m0UUWDUWkzRRtFFg1DNLSbRRj3osGotJmjHvRtFFg1DNLSbRRRYNRaTNFG0UWDUM0tJtFGPeiwai0maMe9G0UWDUM0Zo2ijaKLBqN3H3pQTS0tD3Kd7DW+6a+e/iz/yMkv1r6Db7pr58+LH/IxTfWvNxmx7WW35tTi2+8PpSUdx9KK+fPsegUUUUkIKKKKYBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAK3WjuaG60dzUy2E+hJa/6+L/e/rX1Nov/ACB4f90V8s2v+vi/3v619T6H/wAgeH/dFe3gT57Nd0XKKU9qF617TPmOZiojSEIvUmuq1P4c6vo+gpq09pJFasNwkbGMVz+nf8fkI3YG8fzr7s1r4ay+P/grp1pZeXG5gGWb8aynPlOyjTVSLb6HwP1XPaj0rt/iV8OLz4d332W5dHbP8Nc1ouhXniLUYLKzjeaaZgoC9qfNpcxcHexDY6Ld6nL5dtbyTP12oM1Xmt5LeVo3Qo69QwxX6Kfs+/s82fg/RUudTgjmu5U5RhkivIv2oP2dzpdxNrujW++3OWkWLqtc8MRGU+Q7ZYKUYc6Pkfocd6KdJGY22HJZTzu6im11nltNbhRRRTEFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFJQFhaKKKBWfUKKSihb6jsLRRRQLQKKKKBhRRRQAUUUUAFFFFHUT2GN9018+fFj/kYZvrX0I33Wr57+LH/IxT/WvLxnwnt5Z8ZxXp9KKPT6UV4C1PsugUUUUCCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKkWFmXIBx6gUAR0UYxmigAooooAKKKKACiiigAoop4hkbkRsR/umgBlFKylTggg+4pKACiiigAooooAKKKKACiiigAooooAKKd5bDqpH1FNoAU47UlFFABRRTl5NADaKVuPemZ5oAdRRRQAUUUUAFFFKF3UAJRT/LY9KZ0oAKKKKACiiigAopcH0pKACiiigAooooAKKKKACjnn0opzRnHHWgBtFOWFm4HJpxt5F6xt+RoAjooooAKKKKACiiigAooooAKKcsbN91SfoKGidfvIw+ooAbRRRQAUUuD6UlABRRRQAUUUUAFFLtLHjpStGe1ADaUfpSUvY0AIfbpRTkG5TkU0ZxnFABRRRQAUUUUAFL296MGkoAKKd5bYztOPXFNoAKKKKACiiigBW60dzQetB61MthPoSWv+vi/wB7+tfU+h/8geD/AHBXyzaDNxF/vD+dfUmhn/iWQj/ZFe3gT57Nd0Xu+KVetI33qVeteyz5ZFvTc/bYjngMOPxr7T+IHiS+8N/ArT7nT5zBJ5AOVr4pspBHcxueisD+tfbVjZ6b8W/g7aaRb6hFDdJEFKsa56iuz08LK0JHxx4k8Ual4kujJqEzTP8A3ya1PAvjuTwLeC8hjWW4UfIW7GrXxL+Hc3gHVjZSzrKP7ynNcOw25wMj1rVR905XLXQ+7P2S/i5rnxC1HVW1OZpEjGUGenArzP4xftBa74b8a6vpJbzbJmK7G6Y4rU/YPX/TNa4JO3/CvFv2is/8LP1XOV/emvOp0l7Vs9apWl7GxwWrXw1TUJ7kJs3tnFUqT3IJ560terp0Pn1fqFFFFAwooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigApKWko16AL9aUL82DSL14r0f4Z+E9L8RMVu3Al7ZrmxWIjhYczO/B4OeOqqmjznaemDn6U9beRukbEfSvp21+EejxcmMN7gVal8C6RZ2jlbbOAewr5V8R0pTUEfoK4Ir+ydVvRHypzlgRgij0rY8UQRwa1OsIwgJrGDbua+to1FWgpo/OcTRVGo6b3QtFFFbHKFFFFABRRRQAUUUUdRPYa33Wr57+LH/IxT/WvoRvutXz38WP+Rin+teXjPhPby34zivT6UUen0orwIn2XQKKKKBBRTfMWigB1FFFABRRRQAh6U7bjDUm3dxSc7QPxoAmit5Z93loWPX5QSalbTbqNSWtpQGbhthr6T/Y18J6R4l1O8fVbYXEaAkKR7CvoDQIfAvi3R9XJ0dI007OSQMnGP8aAPzqudNubNQZoJUU/dZxSQ6fcXWDBFLIO+1a+x/ihZeGviJ8I77V9NsEs5rIEgAckA4riPhP8RPDXhfw5BEnhybVNQAw7KgI/nQB83z2M9rjzonX08xSKhbJbbkZ9e1fe+l+EtD+O/hm+M2gPo95EhKMygAfzr5q+F/waXxv8ThoDkG3imKOcelAHk1vpN3dZaK3mcD+JF4NQTW8ttJsmjkVvRxivt3xt468BfAu+XQLfRVv7iD5XbA6/nXgPxx8beF/GSwX2iWf2Odvvx4HHX0oA8hitZZ2IjVnI/hUZokhaE7HXy26kMDX0t+xbo/hjWdU1H/hJGh2A/L5n4Vx/7U1jo2l/Eq6ttCMb2m/5fL6UAeLc9QCFpFPmHIya+iPg7+yze/EbQxql7dR2FtIMoZuM1xfxk+Bmp/CW8VZF+0Wkh+SaMfKRQB5XRWtfeF9S0/T4r64tJIbWXlJGHDVk9KACvqX9n/4WaN4q+Fus6leRLJPFEWXI+lfLX0r7h/ZMsZr74Q61BAhklkhK7R17UAfGOqacf7Unt7dGIEhACjmqM0Ekcm145EYcBMcmvs7RvgzoXwp0G78U+Kwnn/M6W7Dk14FD410LWPiKl/c2GLJZMrCo6igDzZdFvnjDrayBO7bDiq00LRSfOAhHHTBr7K034w6BcNFZWfgu4ltJCFMhjHT865z9qX4OaTo/h6z8TaZB9kS4Te0JGCOtAHy1DazT7tkbOF5JUVPBo95cxlo7WeVBz8q9K+of2K/AGh+NZNYXWYVkjjXhmH0rp/Fnxe+H3wv1640O10VbpVYq0gAx/OgD4vMMgkZXRg442sORU/8AZN3tEhtpgp6nYcV9O/DPwBoXxl+JdzqVvbi10mFt8isOAMZ7V2l18Yfh7Z+KB4WGhq1tv8nzMDI9+tAHxLkwzKZFYDPIFfW3wAvvAXjdLHQbnSd+ouu0zECuI/as+EFp4J1K21LSVH2G8HmKAOAOf8Kzv2Px/wAXSsB1G7t24NAHP/tB+GrTwr8Qb6ws08uFHwMV5r9jmkjMqQvIOuccV6/+1V/yVS+6nMp/lXpf7NvhXQviJ8PdWsJrZG1CGI7GxznigD5R25lCAZYnBWpHs5bdsyRuoPTcOK9Q+H/wwk8QfFeHQZY9qx3G1iR/OvQf2pPDOn2fiCz0Hw7ZbriEBH8ock4NAHzSynaB0OaRV8xQfXrXXj4W+KDjGj3Of92opPhn4kt4xI+lXCIBzlaAObWzlKs4jZkxxgZqRtHvYrdZWglWPqCUOK+zv2ffAvhab4U3Wq63bK7QJuORzkY4rb+G/i7wL8W7i+8MpoqxeWDGkhAoA+EIYJLhiIomdmPO0dKZIhjO05XPHI719J/Dvwzofhf45XXhzUYFltWlKKG7Vyf7QXwtPg34hNaQQ7bW4k/dbRwc+lAHjZt5hEspicrj7wHFXtEtRNqUHmK32fcA7bcgDNfUXxA8FaF8N/gfZCe2Q6tdQZ+YfN3ry/4KeP8AQfD10LPVNLa8+0SBcjHfHqaALnxx0/wra+H9KfQ2BuWQGYqpHr7V4htJVRyDng9a+t/2wvDOi6P4f0W902zFsJo93b3rzH9l/wCG9h8RvGkVvflWt0bO1u9AHkaaHfTIZEtJ9g6naefeqMimNipJQjqjA19wfEL4neH/AIa61LoQ8Iyy20PyGVUGCPXrXy98UPEWja14mS90y0a1tycuhA4oA4lNMurhUKQSunXKqaJNNuo0LvbSInUllNfRngH44eHILK004eG3u7kgJlVU7j+de3fGCx8H6H8If7QvdNjsdSuIN0cLAbwaAPhfwz4ck13VYLdFfbM4UlVPFelfHT4EyfC02Jts3QuFDHap9PpW/wDAT4oaD4d1KGzvdKN1LM4Ak445+tfSH7Snxe0Dwj/ZY1DRmvPMQFTxxwfegD87pIZYWKsux8/dYEdqkt9PubsF4YJGQcZRSRXp/l2vxg+KFvbabaCzhuJQAh/+tX0p481fwN+zra2elHS01G7kUCTAHB/HFAHw1LbSQZSZGVl6hhikhhkuGxDG0nqoFfXnxg+HGgePvhn/AMJpoFutrL5e+SLHI9uK8+/ZJ8K6b4m+IEdrqUKyw52srCgDwiDS7u6aRYoJGweQqGpLfSLy7vltYrdzMzbRHjBr7c+JHjz4dfBXxBLYW2jLeS7sPtA/xrz7wv4y8HeMvi9pV7Bax2ULSDcj4wOtADfhd+yvq+reHL241GwaKR0zFkjJ6e9eK/En4R678Opj/alrJAjMdhyOf1r728XQeKbJtYvNNzJp6pmBYzx0HSvhT4qeOvE3iGb7JryP+6YqvmA5HWgDzjceBnb7HqaKaB8xycfyNOoAUAHryPSuu+GXgub4geKLXTYFLCVgrewrkV9vvDmvsH9lPwnZ+FfCd/4yv0CFIy8W4YPagDyn9ov4c6L8OdSt9N02RWuYwA4X1xXjElrLb/fjeIE53MOtd34u8VP48+JD3ly3mCS6/i9M9K9r/ak8BaP4b8FeH7mxtlgkmiDPxyetAHy3HYzzRmWOJ3QdWxTI4zL/AKsbpF6Lg5r6w+A/gLSdd+D2s3l1brJcRwllfHTpWJ+yv8K9N8YeItRur+MTR2kh2w8ZbFAHzr/YN+qsWs52VefuGqTxtHhWJ3+h6ivs7xX8cPD2halc6Vc+EJY7dMxiTYPz618p+MLzT9W8UXM9hCYoJG4XFAGCtrNIA3ksVbo23rU82jXlvH5klvIqKM5KHFfdXwu+HPg5PgXZ69q1pGZIot5bHJrzrxB+0B4D1LTLzRxoHlqqlEmwv+NAHyaWO7hME9a90/Zz+CcPxGurm81IiLToAWZmHGMV43bW51bVhHboSJZNqqPc19narND8CfgOloFVNXvrfDdiM9/0oA+a75vD3hX4jurILnTIpcEDoa+ofDfhnwH8Tvh7quo6Xpqwz28RIbA68f418MX0r3V5JO78tljj1zX2d+yiS3wi8SEnH7g/XtQB8c61bi11S4jVtoRyAPxqG3s5bpgsKM57qik1o6rb/avEVzGP4pyOe3NfYXhfwT4X+CPwytvE+t2i3l9cRCRIiBk/5xQB8W3FhcWufOgkh443riolXIXaoJ/iHrX3NoNn4K/aS8N6gtppq2WoQodowM569vrXz38L/gjJ4s+KjaJP8ttBLtc+woA8og0a8uthitndTzuVTUElpPbzGKSNhJnIyMH9a+1fHPxG8CfBW/XQ7HR01Bo/kkkAH+Irwb4z+MvC3jS+0680iyFmwYGVcD39KAPKINJvLoFoLaVzjAO0mmT2U1q2yaF4/TcuK+sPBvxY8MaHo0Nvp3hOa+lCgM6oOT69a6Tx14B0T4mfDG98QwaO+kXluhfbIoBP5fWgD52+CfibwzoN83/CRWIvFc4GQOOle3ftBfD7wxF8MLPxHolmtuLiPfgD3NfJdupi1GNP4hJtP519jfGLH/DNOhgk/wDHvjHvk0AfFrspb1was2FjJf3NvFGmWkbGB71WY7WJIxjvXtn7Lfw3fx940gmmXNlbHe7EccUAdP46+COi+A/hDbatfyKmqzx7kQ9a+cFge6cpHG0nOPk5Ne+ftZfEhvEni06Lac2NmfKRQeOK734F/C3w94T+HMvjbxDCtwqx+YsRHU0AfJtxo93ahXmtplU9C61Vt43uOEXex4wASa+6PBPiTwJ+0BJdaK2krp9woKwtgc/zryn4YfCuz0P9oJfD1/D5tos5UBhwRigD52TSrufKx28znGSFU1BJBJDN86MpB5VhzX3r8TPGXw/+EvixNKXR0ndjtkKgcV5p+0V8NNETw7p3jPQ4FS3mxK8Sj60AeV/Af4MyfFDXmsJd0CL0LqcHjPpXKeP/AALc+EfEV1p6RvIIGK7wpxX2L+yf8WNB1rWIrK10g21xGmGmOPT2Ncf8a/jl4Yttc1TTm0LdKSy+bgdfzoA+PXXqM/N0Ndf8LvAM/wARPFlppcSk+Y4VmA7Vy104uLp3VdisxYe3tX1p+y54Xi8D+Eb7xtfJs8uMvCW4JNAHmP7Rnw50P4d6nb6dpLrJOvEm3+9ivG/7MumYfuZMngAKea7jxF4pm8e/EJry6bcJbnhSc96+w/iFb+CPhJ4V0nVLrTFubp4gyooHPB9aAPgKfT57NgJomjzxh1IqMW0zR7hE5TruxxX2trXhjwn8efhffaxpVoLLUbdN5jwM/pXM/A/wTonxA+H+r6U9vGdTtUZVOOeMUAfJcbLNlQOemKn+w3CmMtDIcnpivUvhf8KX8QfE6PRLiErHDPtfjt716d8cNQ8NfD74kWdraaektvZsBIi4w3FAHEeDfgG/iD4b6h4jdjFLBHvEZU57e1eO3el3FpI6ywPGitgkqRxX6JfDz4saDq3wg1fUodGKW0MW5oePmHHvXyp8WvjJoPijT5bKy0NrKQkjzDt/oaAOp8N/DHSNQ/Z5u9eeEfbVizuA5PIr5uutJvYdztbyqh/iZTivu79n/wDsofs43Fxq2DbRx7ijdxmqvw68T+DPjTLeeG4tFW3ZMpFLgc8detAHwdu/eFQVDE9cGrcej3c67o7aUgchghxXvfhH4BRa58ZrnRGP+h28+G44A61654s+Jnw/+G3iQeGI9IW4SMiF5AB+dAHw7LmJjuBBzyCMVMLKZsFYpGDfc2ivqH9pj4P6Rpum6b4j0OFUs7wBm2jpkmvULHwr4I8HfBnSvEOpWKSzrCH6cnrQB8JzaVeWse6W3mhHXc69aqN8pU/wnk195aDJ4P8Aj94H1QW2krY3NpEdrEDn8vrXw1q2ntY6lc22OI5CoHtTewvtHR/DHwTdeOPE1tYwKWbdzgdq+jP7IbRcWzfwfKat/so+FLTwd4RvvGGqIIzszEW69qTUNS/tS4lnzkOxYV7ODPn8z2KW35jS7aUfd/nRXsHyon3e5Fa+meKtT0gYtLp4x6A1kUHOOtA4ysaF9rV7r1y0l3cNI/uapNnkHpmksvvtwM05+rZpl81z6y/YO/4/NZ/3f8K8V/aK/wCSn6r/ANdTXtP7Bv8Ax+61/u//ABNeLftFf8lP1X/rqa4qf8dnp1P93R5p/jRR/jRXYtjyOrCiiimAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUn05o16Bbr2FHuM1t+F7zULTUI3sVZmz91azLGykvrhYol3MTivQ9O0t/h3NDeXCeZuGSK8vHVFKHsz6HK6M/aLEJWSPW/BPjIahGttdHy514Kt1rodelEOl3T5wNh5r5r1bx676817ZgxEnO2u9k+KkGqeFJIppNs5THNfA1cml7WNWKP1rDcUUfq06E3qeO61IbjUp2zxuNUQAOlPuJPMlduuTTOwr9Kw6caMYvofiOLk54ic31Ciiiug4gooooAKKKKACiiijqJ7CNkKR614L8UtKvJ/EErRW0zru++q5Fe9/xAnoKf/wALy8K+EVbS9R0T7VK3yvIQP8a8zGfCe5lnxnyJLG8TGNgRLnkEVcXQr6RGdbSXCjhthwa+lPg/4J8MfFz4tTXEcKpYht/kNjJ9q7zx98WPDnw+1afQ28IyNbxNs8wIOR69a8BH2PRHxDNGYNqSDaR1GMU6FTdMqxjIb5a674p6xpmv+JprvS7f7NbOc7GrY+Afw9Pjzx1Y2Zx5CuGkyDikI1dP+B+o3Wn20xjYGSJXPHqAaK+5Wm8N6axtA0RFufKHI/h4/pRQB+XlFFFABRRRQAq/eFJ/d+lGcUfw+9AH1l+w7/x8apkZHlt/IV13wnmZvD/jcBeMvz+VfO/wV+Mx+EslyywtL56lfl7cVr+Ff2gP+Ed07XbX7M5XUA3ORxnHv7UAeueA7MXvwO15JCWLAjP/AAIV0OoR6X8EfhRYanp+li9vLiPcXC555/wr500P47SaT4Fv9BS3c+dn5sj1B9a734f/ALVllaeF4dC8Q6a1/bou0F8HHt1oA97/AGcfiZrXjzR9UuLvS2sYRGxB24yMD3ryT9nXVIIPjzfxyusTSXDBd30qC3/bTstEguLHSNEa3tHBTC4Hb61892fxGutN8atr1iWhn83zSM80Adx+1J4d1XTvinqE08EzxySEo6rkGvJL7Sbyxt457i3eCFxxvHWvqG2/a40PXrONPEWgG7uox/rTt5P515R8ZPi5Y/ELyotP037DbRdF4/oaAK3wQ+FniL4jXFwmhSvEY/vMhrH+KXgzVvBHiqSy1eVmuo35Z+tdV+z98eD8Eby8nW1a5E3Py4x29T7VgfG34nf8LZ8WXGr+SYfMbdtJFAG9qH7Qmr/8Ibp+hWG+2EAClozj1r1r4a/Hrwt4y8NxaP4+bf5ACLLJyT3r5Ix7Um3AAHagD3/9pL4o+HfEdvaaL4XjU6bajaGX8f8AGvn7YRwOlPpCwXigBoVs194/sY64ug/DLWL5k3GKLOPyr4Q3CvZvhj+0C3gHwfqOirbswuk2Bvy9/agD6i1/XNO/aZ+Ht9ZxyLHqsCsFh7nBr59/Zz+Eaah8VH0vXINogl2+XIOuK84+HPxc1D4feKjqtuX8pn3PGp613V9+0s0fxCh8TWFm1tKGy4GPm/WgD1n4kfGWfwd46Ph/Q9B/dxyCMMsfH866L9rK7ub/AODFhNeQmGaSEMV9OtcJf/tdeGbxvt8vhxn1PG4yNtyT69a4X41/tRSfFTwzHprWLwKFwM4/xoA7j9iO4+z2HiVhyBGf5LXzV8TpmufGeoM8mS0zZzXYfBT40N8KbPUYDC0wvAQT+A9/avO9f1T+3NYnvGG3zHLYoA+rv2HLqGTTNdsldRdTIdgPU8CvDtR8J6t/wtZrb7PN9pa6wRt461j/AAt+Jd98MfEMN/ZlsK2SqnqPSvoVv2wfD00x1H/hHd2pDnzlC9fXrQBf/bCuE0zwL4fspCPtYiG5T1715B+yNdRQfFPThI4j3PyT9DXJ/F74s33xV11724DLEG/doTworm/CPiO48H69BqlsxWSJgw29aAPV/wBq7SLyL4nXkxgkCb8qyqSDxWd+zN49k8D+OrVHk/0e4cLIvTr613et/tZ6X4l0wx6poBubtU2mXA5OOvWvGPCXh3WfF/iZbjR7KR2aYMAo+7zQB9/D4X6b4D8Rar45ZY1tnQzI3ocCviK6+M99a/Eq715QbhhKWAc5FfU37S/xAn8J/Be00O4k2ajNbhZEB5zXwGxLMSeSeaAPpP8A4bP1jq1jECTgcf8A16o6v+13quq2s9o2nxojDaGA9fxr56HzYXtSr94c7h2oA+0Ph9fNd/s465cbsO0Jb9RXAfsatn4lEnA+c5rjvD/x2Og/DS78LC3Z1mj2lgfpx1rC+D/xTb4YeJJNTWB3yd2MigDqfjhq02gfHm8vY2/1dxkMOvavrmPwTZfHHS/DniGNEdbdVaZjyeM18GeONeu/il4ym1K1tZHnuHyI1619i/BLUtV+EfwP1C81lZLcmA+VHKQCOnSgDxD9sLxoureMjplm++1sz5aqDxgD/wCvXh3hKGZtesNqyFfNXkKfWpPFniCbxNr95fTMxaSQ8tXqfwz+NWg+BdLhtbnQzeTRj/WYHP60AetftiWs0ngfw0yQs2IRnapPrXzZ8N/Fmu+CddjvtIhkeeNssig5r6C179srRvElnDBqHh+SZIhtTdtOP1rzLw/8cdJ0DxtNrEWj5tJGz5JA/wAaAPdPAPx90f4napBpPiTw6zXc2EaTZ/PmvJv2sPhpp3gTxdENJHlxXJyIl967W2/aq8H6WTf2vhrbftyGULwfzrw34mfGC6+InixdVuUcqr7ljY5xQB7l+zj8JLXwrocvjHxLH5Vuq+ZEJFzn0rxn44/FLU/iR4iuH8uaPT0bbGmCAB6163Yftj2MfhW10S50SSWBFClflwf1rlPFnx/8L+INHubOHw4beeQYEhC8fkaAPI/ABx4w07LBh5q49ua+jf22LOeaPQZY7dpotgJaNc9jXyta6g9lqSXdu67o5N4XvX0jpf7WtneeG7ew1zSDfPbpsDNg/wBaAPOf2bLpdM+LmkST4VBIP9YCDXfftmeG9Uk+IjXIhlmtZWDRFBkdK8i8ZfEGLUvFyavpFr9jVW3IgwMfrXumk/tfabf6XaweItFN9NbqAJGwc4/GgDt/BsL+G/2Ybo6l+5aW3wqScE815d+x2274nOw7OT+lc38aP2jbn4h6ZFpenwGw05RsEfbH4GuS+DfxVb4XeIn1FYmmOP4aANH9o64a6+JGpO53HzD8tcH4X0/UrzVAmlIz3GcjZ1FXfiB4tHjTxNNqgXZ5pJK+tTfDnxs3gXXotRih8wo2SKAPqr4C/FrxZa6PqFvqsEj/AGRdojkU5bpUv7R2j6b4o+EC+JZLCPT9S27tuMM3JrW+GP7RWkeI9J1G9udISKSJMkcfOePevnn43fH6++JW7T442tbKNioj4xigDxjacdOPSm7T6U/IDY6nuaGyFNAHQ+AfCdz4x8S2em20ZeSWQDj0r6g/aS8RR/DLwDp/gzT2CSGERzeXwc8muJ/ZJm0HQNTutb1e5SOW1+aJH7nArzj43+Pn8f8AjO+1AOGjMh2AmgDivDsm3WbNn6mdSSe1fYX7XFlPffDTw1PbRPcIsK/NGM+tfF8E/wBnnSReqnJr6g8F/tY6fbeEYNF17Tf7RSJdqlsHH60Ad7+zzpN3Y/ArWpZ4JId8BIDjHpXzp8L/AIla/wDDvxJd6jpNtNNEsh8xIxwRmvVdY/bCim8M3mh2OmG3tZUKLjGAPzry/wCEHxmsfAV5eC/0v7ZBO2SMDv8AjQB9M/Db4n+Gfjtff2Lq/h/7PdTfL5jJ1OK+XPj/AOCLTwF8RrrTdPK/Z0lyu3tXsq/tXeF/D0TXGhaB9mvSOHwOD+dfOHjnxpc+N/EE2rXefNlbd81AH1xeXUlr+yOBG+MW/wCfNfEUgLyP6bsmvZJf2hHm+FI8JfZWO2LZuGMdfrXji/vJlB43etAHvP7KXw4TxZ4ujurqMfY7M75JGHAxzTP2pviOvjPxhNZ20u6xtm2JtPGBXovgbxdoHwr+Btw1rdxyaveQYZV+9n/Ir5M1C8N5eSzO25nYt+tAFZl3Kwxz0FfZ37JrC4+FniS3j2yzGEgJjkdK+Mzz0r0v4M/HDUfhJfyGBGnt5R88Y6GgDlr+ymsfF7i4ikTbc5O5SAOa+t/2lrC4174I+HrvTSZLaGAeYYxnHWvFvid8edI8d6XILbRPst5IMNNxx79a0PhX+08/hPQX0LW7FtS09htCnBAH4mgDvf2HtJu7GfVr+dHhtlQ5kkGAeBV74E69Zt8eNWAK4llYI2evFcf4o/azto9Cn03w1pL6csylWYYGf1rwTw3421Hwz4kGt20rLcq+/ANAHb/tCeF9W0v4kagbyGZkkkLI20nIrk/CPg+7vvEml219byQW11IAGcYBr6Etf2tdC1q1ik8RaH9ru4x94gHP615d8UvjZb+MNXs7vSrD+z47R90ajHPXjr70AfRnxh8SWfwJ0/TNN0bQhcTPEoMyJnP610Wg+JdY8YfAXW7/AFSw+xCSEsnGM9K8k039rjRdU0O1h8RaMb27t0Chmwc/rVLxF+2Kmq+FL7QbfTXtraZdiLxgD86APmiTC6wd/Xzf619jfFK1m1b9mXRmtUa4CQciPkjrXxfLIXuZJB8pZtxr3f4YftNy+DPC66JqFi2oWirt8tsYx+dAHg/2Wb7QsTJIjs2ArDv6V9ofD+C3+CPwJn1m5TyNUvYMqvRs/wCRXlGl+KfDXxM+JWlyy2i6VZJKGkV8YPX0zWp+1p8ULLW7y20PSJVextV2jYeO9AHgGqaw+qa1JdzEu8rbix+tfZl1ayeI/wBlFoNObzZ47fDRp1618O7s7cdTXt3wT/aNu/hrZyade2z31jINrR5GMfiaAOo/Y10G+b4jR3UkEiQQsd+8Y7V6FpOpQah+1diAgkTY3D6VzF5+15pGkWNwnh7RTYXNwCDJwME/jXJfsx69ca98dLO+u3/eyzbmZuh4NAFr9rbwzqI+LFxKtvNIssg2sq8V6X8UIm0X9mmws79R9oa3+VW69TW38Yf2itG8O+Obmy1bRPtz27YSQAdfxNfNXxw+PF58UriKFIGtrCHhY8jp+dAHb/sSyInj51JCM+QqnvxXm/7QWl30HxC1Fmt5hiQkMF4rE+GfxEuvhz4mttUtd4MTbtuete1eKv2oNB8W2U32jw9m8ZceZgdfXrQB4N8P/DNx4w8TWVhFEzyNIAwxX1V+0r4htPh/8PbDwbp8gjkEISZV4weTXIfss3Hh3Q9Sv/EWqTxRNGS8cb+vavH/AIx+PJvHnjfUL95C0LSHZk9qAOd8Kjd4ks+ekyke/NfXv7Xui3dz4D0CeCGSaJYV3FBnHWvkDwnKI/EliST8sij9a/QH4w/F7T/h/wCENGttQ0038EsK5HHofU0Aebfsq6XdaL8L/EN7qKtBZvCdpkGM9K8r/Z18eDwj8WCkp22dzOUfnjBrS+Jn7VQ1zwq2haLpzaZZyDaQuOfyNeTfDfwzrPiTxBBJp1rNPIZASydBz1oA+94/h/p/w78Q6/40fy0tWUyxH6gV+fvxF8VT+KPFt7fSsZC8pxzX2J+1B46uPC3wq07Qp5f9NngCyLnnvwfyr4RLMzAnGSf1oA+xvgjBJdfs76/BGrPOYT8g57ivkTVLO6h1CY3FvJGVcg7lIr1j4L/tAXHwqgmtnga8sphhouMEfnU/xN+NWheNNJnitdD+yXEhyJMD/GgD2PwTO0P7JN8qndiDH6iuM/YoYf8ACzCQMHcf5Vw2j/HptJ+Ft14SW0ZkkTbuyP8AGsX4K/Fl/hX4mGqCFrjdn5eMigD6t+GOqW0f7Q3iCBnVJpJGUK3fgV82/HTw1qdv8Xr6MwTM01x8uF47Viah8Y71fiM3iqzV4ZZJN+3Ne4f8NcaBqHlXepaD9p1CMDMrbeT69aAOr+NH/Ej/AGftEsL1cXrRLhW69TWd8Wrhl/Zt0cNwPs/I/E14T8afjtefFTUoSY2gtIWykYPAH51J4r+PDeJvh3beG2gZBDHs3cUAeu/sY7f+Ee8SAdfKb+QrwHSfC9x40+J39nQRs7yXRHA461t/B342H4W6dqVuLdpjdKR8vvj39q9G/Zk1jQNH1bUPE2rzRx3ClpI436j2pvYNnc6f9pLxNbfDvwtpHg3T3Cy7FSYJ+NYWjMW0+H/dFeK/F3xo/wAQPH1zqBdni875MntXtGh/8g+3AGPkFezgj57NNNDR6Liloor2D5UKSlpKYIdZ/wCsNOfvTbL/AFhp0n8VBZ9YfsGf8futf7v+FeLftFf8lO1X/rqa9p/YM/4/da/3f8K8W/aK/wCSnar/ANdTXFT/AI7PUqf7ujzX/Gij/GiuxbHkdWFFFFMAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiijcAqWG3eaRQoZh7VLYabPqUmyBDIx7CvRfB8Gn6Fa3EWrIqT7cAMK4MRiVTXLHdnr4PBOs+eT91C2fhWz0zw6uprMouQNwA6g1yPiTxddeIFSGVyUXjNQ6zqz/AGiaKCQi3ZvlUHisMkAY7VzYfDzk+arqduLxzhD2FDRDdnYdfWn5KjYKSkxXrKMVpY+evbYKWiirv0QnJy3CiiikSFFFFABRRRQAUCiijqJ7CHoa+ffipz4nnPHDV9BN9018+/FTnxNN2G6vMxnwnuZZ8ZY+EfjTXPBGvjUdEillljO5ljFfU3w/+N+hfFrVIdK8QaAReTfK0rx96+afgr8UrX4Z6m815ZLexv1UgdPzr2iP9q/wnpbC8sPDXl3g5DKBwfzrwEfY9EeeftSfDTTPAfjaS1011+zu+Qi/SvU/2f8AQ4fhb8MNS8Y6iBDcSRFodw57CvD9Q8byfF/4mQXmqS+RA8oP7zsK9X/ah+JGlQ+E9P8AC+gXMb20cYWTy+nekI8jvvi7f3F9cSi4bDyM3X1JNFeYLnAyOaKAI6KKKACiiigANH8XqKKKAAEsTnH404DgZ24PUUnbgkH1FW49GvZIfPFpJs/vbDzQBU/iI49vekKr05B9V7GpZLaaJEeRJY19XXAqIZOeh560AKxO3JOSo/OlVWkXOMkflTa9d/Zr+HNj8SfHlvp+oDEJbB9DQB5LHC7Z+RT+Bpr/ALsZZQB22195+JPg58HfCGotY6hqltb3Q4dHB4/SvDPjv4a+Hmj6Up8K39vdXHcRg/4UAeAcnvxjNN5bB+XngZ71NNZSx8vC0Y67nB5pI7WWZWkjRnX1VSQKAIvX260envU8djNNu8qNmYfeZVNWV0HUJF81bW4KgZ3hOKAM/sT2FHrSn5cq27cDgg1fttDvbxlaO2klXvtQ8UAVBCeRtbj2qPachMnHUGvqV/hbpEP7Ov8Abz25XUzDncRgg5r5jttNuL6TbBDJNzjIXNAFXcWIG7Az1oViVySzYHQ1eudDvbH/AI+LaSNQOGKEVXjtZLhsRRM8nbAJoAhDY917hqThdxAwe2OtXm0a+jVWe1lVT/EVODUd1p91Ypunt3gVum9TjHrQBXA5GWJBpu5eMjJap7ewmusrBEZXHI8tSamvdJvbGMGeCSNT/wA9FOKAKu1lOFbBpvKjK49M+tSw2z3Em2CLecc7Qasx6LfTpuS1m2jsiGgCkuMFeTtpyktgjP8Au09rORZmjaPbJnG0g5r234F/AdviRY3s13E0JgjJG9Tz09qAPDDnnOSD/C1ep/Cn44XvwrgmFnbpI8gyHI5X/OK4nxF4ZvNF1K6tzbSqInKhmXjrWTFay3UgjhVnfphFJOaAOu+J3xS1f4maj9s1KRpHY8LngVxSqenU1fuNH1C1QvLaXA/ulk4pNNuI7W+R5Y1eNWBZO5FAFRo5I+sbf98mmupXKjg9RX258DbLwF8VLGayXQzFcwx/NIwGM4+tfJfxQ02HR/F2p21smyKOUqv0oA5crub7oxnmhcDtkkYIp8NrJcylIYmlbOMKMmrsnh/Uoo9zWk8cf95koA1fAvjaXwTrkOoRxec8T5APSu5+KH7RWvfEbT/sEz/Z7Tbt8teleQgFmK4wy9c1f8P6LceINYt7O2jaaaRgoVeaAKjKwycZzUYUk/ID9TX0J8cPhRofwz8I6SNw/teaMeYncHmtH4B/s82vjDR21/xDILLS1XeHk6MKAPmxYznaAVA70gjZjhQW44HvX2nfeAfgxfW1xY2mqW8d6ikKeeT+VcT+zv8ACvQfEHjTWdPuVW7tYC3lv17CgD5hYBWJYEH/AGugoDbv9k/7Ndt8SvC8tn4s1G2srGTyI5CFYIcVx0NlK10tuwdZM424xQBEq7lxjGOjUudsijcwr7E+Df7JOj6xY2l5r97BA84BEMh5/QV5N+098L9O+Fvi59O005iH8Q6UAeJgffKhcjr600YZgQW644r1n4F/BhvitfyCa5jtLSM/O8hxxxXv/wAUP2W/C3g/4bTanYXEdzNGmfMToT+VAHxTj5sDaeP4u1GfmwPlHf0p00YjmdV6KTTKAFwZGIOMdqMrGARlc0LgfXoK734N+BF+IHjCy0pxlHcbjigDgVb5i3r/AA0+NSwJHP4V9eftMfsx6R4B8Gx6nowG6NN0m0V8veE76y07V7ea9i863U/OnqKAILHVdS0e3kihMsSPw20YrNZjKzFiWbPJNfcHhPQvA3xL+HurXun6V9muLeEkMQOuBXxPqMXlahPGPuq5A/OgCr95eOB3pfvZHbHFWbfTrm+b/R7eSfH9xc0Npl3HIY2t5Q2ehQ0ARW+pXFntEchVD1wetQuxkUFgCT+lGw+ZhVbrjCirbaHe+W0v2SXaf4ihoAp58sZpWyx+8APahUMchUrsI6g9aljsZpozMiSOqc5AyKAIMDkjp70Kc84x2+Wry6FqLxiX7FMytzuCHAFQR20ssm0BpCP4QOaAI9rk/wARx6jpTS3YjJU8V75o+m+EV+ENybuNYtbVPlDock/lXhK2zTXGyFN5ySAoJoAj2sxTGRu/u01d3Xv0ya+gfAv7Oo8RfC678Qyh0ngj3hCpz/KvDL3SLqxkkMlvIkYJ5kXAoAge8u/JMZLuijj0NV8eoXP94d6+pfBPwt0fVP2f5Neltg98IiwOMmvmy50K+haWU20kUStjOw0AUulN53feOR2qa1tZbxm8mIyEddoORU0miXiqu+0mC9mZTQBU53ds9qPu8/pVm4025tkBnhkjTtvXGaZbWdxeyeXBE7P22rmgCFmwM5I9hTSSvp83XNXrrR72xXfPbSjsTsNU+CuBt/GgBm8LzlhzjNP54bJ4rsvhb4Dl8feLLXT4ozJvcLx0rtf2kPAOhfDfU7bStNKtcxjE59DQB4xzz3J55pQj7dx3HjsOBVm10m7vhugt3uFUfeRSa+iPhf8AC/TdU+C+u6rfWhjvbeIlS689RQB829Op/GjpnOTzUsyFruSKPJOTjAz3qy2h36xCVrS4VByW2HBoAqwzS2+Hik2MoyMdajuriW6zJK3mN3J61JBbyTy7YkyScEKpzUy6Lesnmizm2nqzIcUAUdo29Oe1Hl/JgDd3BNXZtLuYIBK9syJjIbaRTI7SaaXZGrO/91RmgCsS2WznOOMV1Pw98e3vw58RQaxZEtNFzt9TWLdaPfWkYea3ljB7OtUW2rnBAIFAHR+PfHN14+8QXGr3hKyzHJU9Qa57buAfaB/vVreEfDsniTWbexhQyySsBha9o+Pnwp0H4Y+HNNjRgdVZBvQdQeaAPADxnkjscUo4wCM+nrU9rY3N4QsUTzSeiLmn3Ol3llzPbTRD+9InFADY9RuLeF44nZVb+FTVTmTG4A54JqeG1luuYkeTHOFFN8l1Yoch+m0igB9jdPaXUM6nDxyBgPXFd/8AEv41ap8RrKwtbtMRW67Bn2zXAtZzLImYWVxwF2nmrH9h3yqWa1kUDktsOBQBQY9VPTPAFenfC/44ah8KbWZNPt0eRjxK3bpXnZ0278kzG2kKD+PYeap7/wB4Rnhv0oA7L4kfE/VviTqhvNTkZ2DcLniuN2jLAc96eSOT3ApYo2k2oqlnbsKAGd+QzN+lIVPGQpz1NX49FvZ5PKS2m3r12qc1BeWdxYlluI2hftvXBoArKp3bTjBoUHbldybT0HU1Ju5XnINH3f8AdHWgBCpI4Y9P4utN2hscZB6mpord5m2xRmRieVAzVqLR72Zv3drO+zqqoaAM7lhxu9qcdxbuBmpzayxSGNo2Rs/dxg5p0llcQj95HJGT0MgwKAK33cls5IwCtWo9QmhQpG7kN/dqOG2e4k2RxtK3oozX0R+yb8MtL8catfwatbmUxA4Vx0OBTewS6Hz3at5t3EBktvHGK+n9HUrY24Iwdgpvw9+F2i6h8Z9R0uSFTbRMQq46Yro9Y0+Kx1SWCIbY0baq17OCPns13M6ig9xRXsM+VCkpaKYIW1/dyEtwKdJ/EaZTm+6aDSJ9Y/sGf8futf7v+FeLftFf8lO1X/rqa9p/YM/4/da/3f8ACvFv2iv+Snar/wBdTXFT/js9Op/u6PNf8aKP8aK7FseR1YUUUUwCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooGHtU1naveXEcSDlziiCCS4bCKWHcgV6RY+E7C38NLqAnUXSLnb3rhxGJVONobnr4PBOpK89iO10Wb4eTQX8q71cZIPauZ8ZeKf+Ek1AzBNgz2pniDxde6xD5EhykfArnm61z4ehKX72puzqxeLjTXsaOyEZ2Oc85/SjFLSV6rfLoj5/mlLVhS0UUyQooooAKKKKACiiigAooooAKKKKOonsI33TXz58VP8AkZZ/XdX0G33TXz78VP8AkZ5frXmYzY9zLPjOJYkMTnORQG+YkMRx09aR/vD6UlfPrc+x6Ilt7mS3kjkjbZJnOVqWbUJ75i1xIzljzuNVaKAJOPWio6KBBRRRQAUUU6gBtH1peKTcF5PSi4z1j9nf4ZxfEnxta2k6f6Oj5dj0xX0v4p+KXgTwF4kg8JHRo5Y43ETzbR/jXk/7Eup20PjpopZAhkOF3d+K5/44eC9Y/wCF13CLbTOZbkbHC5HamI9j/a+8EeHdN+HunX2jWiQCZN+5RjrmviYrtwMcgV9y/tXafNpPwY0S3uciRIRnP418N7umORSDUQcmvor9iNT/AMLUt13bcv8Ae/CvnYH8K+iP2KSV+KFuc/Lu/pTDU90+OX7Pem+MvGl3fz+JbazkZsmNic/yr5n+KHwp0/wFr2nx22sw6ijyAMsZJ7/Stv8Aax8Taja/Fi+ihu3iiV+NpxXj9lq1zqGrWkk87yFZV+ZjnvQGp9G/tReDdL0H4f8Ah25sbdY5poQWZRgnrV79nLwdpms/CfXbm7tVknjhLBiOR0rW/ass5r/4XeGpYY3lVYBllGfX0rS/Zr0m8svgnrs89s8cclucMwx6UaBqc9+yX4T0PWbfxLNqsCSxW7MRuGTgYrrPBfxb8Ea/4wHhKLRUWKSQxJMyjr+dc7+ym2zw74xbKjCvnP4V4r8GJi3xwtdjfduiDn6mjQNTrfiR8FbdfjqNEsgBbyTg7QOMV7L428b+DfgbLZeHW0uO6l2iORto4qt461CDT/2m7Ez7Yx5o5PfivKP2wvDuoSfEoTxwSSRzuPLZFJHSjQNT3340ahpGofs5S3OjxhbeWDO0dua+evgT8TNA8H6G8VxoEmqX2f4FB5/OvW/Emh3ek/snxwXSMsnk4LHjvVP4W+H9K+HnwUPilNNXUL9o954yc0aBqdX4MvtN+M0dzpWoeE5NNEqHZK6LgfrXEfAr4Y6T4d+Juu2N9bpdwW7ttXHQACu1/Zt+Lmv+OfEkkUulG3tVyCSmMcfWo/ArD/hb3i5ie7cD6CgNTpfDN94H8caLq7jSI4l00k42jnGP8a8/+JemeHvij8H5dU0vTUs54V25A98VB8F2/wCKd8Z8Fh8/Pp0pngaRYv2f9RkZsoDnPp8wpBqXfhr8J9O+Fvw8XXrzSzqt9Km5IlUE/rXQ6Jpek/G7T7rT73w2+kyqp2NIgH9TWh8QviNqmg/CHSdR0C1W+ijiBkCDOOteDab+1Z4yuJHistLk34wdqEYoDU0P2d/hjY2Hxiv9E1SNZoY5T94cAYrvrn4ueAvB/jxvDtvo0cqtL5bSFRj+dcd+y34mvvE3xX1G+v0aO6fcWVvpXjPipi3xml65+19fxoDU9G/af8J6V4C8babrVjCotLlhK0ajtg/4V7/+zd8X9H8XaRfG00oWogj+fgc4A968R/be/wCQf4ZzxmNeR9DV/wDYnbzdN12MMdwjOFPfgUBqYvxg/aB0DV49U0uHQRFd7mUTYHX865D4AeOPD/hSe7m1LSG1O7Y5iUKDzxXmvxE026g8UamJLeQfvGxlDjrX03+yz4O0mz+HN/4muLBbu9hiLqmMnPFPQNTt/BfjjTPiXqsOl3ng6S0tp/lE0kYwP1r5j/aN+Gtt8O/HFzBa4EDOdox0r3/4L/HDxD4w+I1tpsWkNbWiS43FMAD8682/bkbf8QZM9d39KA1Nz9hz/kLap/uk5/AV4H8Zsjx1qxP/AD2ORXu/7DNyja3qUJf94ykKPXgV4r8cNLvLPx9q6T28mfOJB2nGKA1Pev2Tfh7o9x4dv/El7ZrfzwLvWAAEnpVzXv2gPDxa9sNS8JvaxjcisyD0+teQfBX4veI/hdHJLZWs11YPy6KMj6V9I/DjxF4Z/aGt7221Hw9JZXQUkSMgHOPrSDU+ENcvIrjU7ia3j2IznavtX0V+x74Civtal8S3wC2ViPMy44OB/wDXryPxx4MGnfES40azG8edsUDnvX0349WD4HfAuHTLX91qd9BhtvB5/wD1UBqeDfHv4gf8LI+JMrRMXtFm2IO2M44r6U+MUz+Cf2c9NsLBfKMtuPmXj1r4ct73/iZJcMcjzA1fdHjKCT4rfs22k+njzruC3yUU5IphqfC7Xs6TF0kfJ/izX1P+wmzP4q1IzPlW5OfoK+aY/CmrXd9JBDp8xkRsbNh5r6Y/Yhs5dP8AFmqRSxlXQHcrduBSDU9D1X40+C9G+JEvh0aPHcebN5bzFRjJ/GvGv2vvAtl4K8SW2o6aixx3HzqoGMdT/SuN8UMW+O0mSSFvM5Xp1r1r9uxCU0AYyPKXp9DTDU86+AHj7Wda+Imj29zdyvCsgURk8Yrpf238t4+Gc5Pr9K4T9mXSbtviVpcwtpfL80c7eK9F/bb027n8cealvIUXuFJ7UaBqfO2g+LNU8PLJFY3TwpIcFVOK+ztPv5rr9ljdM5kkNv8AefnvXxBY2ct3KFiiZ3DcgLX3Fpul3f8Awy35QhZpfI+5t5oDU+JdP8P33iHUja6fbvcz7j8qDk10v/CkfGm0E6Fc4PT5R/jUfhDxjqfwz8TjURaskkbbgrDFeux/tr+IOMWakL7D/GgNTyG6+Dfi+1iMkuh3KKvO4gcfrXv37GPhL7HqF/rd3GV+xqWyw7gCuV1L9sHxDrlu9obNcTfJ27/jX0h8HZNH+HfwZOqa6VhF9FvlLcHk/wD1qA1MzSvFR+NXgnxTp0hMskQZIwevBFfBmv6fJpOqXEDIVZHII/Gv0M+Dfjb4bf2xPYaHIour0kFc9Sa+Rv2pPA7eEfiVfxL8sUshZfpQGp7L+y0T/wAKs8R7znEJx+Qr5JvLc3GvTRD5C0xGe3Wvrb9k/N18NfElrGd8vlHAHXoK+XJ7G40zxdILmKSMLc5ZnUgdaA1Prrw3ofhn4D/DG11zVrFb69uow4Ugfp+Vdj8M7rwT8XfDOpalFpMcN1HEzFSo4NcX+0Rp9z4m+C+g3emxNcRJCMmPnHWk/Y78MajY+D9cvLiGSOGSJj8wx2FINTz/AOAvwesPGnxC1Ge9AOn2krEqRwAK9Nk+NXgPTfGX/CLjRYmtw/ledtGD29ao/stXkMmveKrMODNIzqseee1fPOseDdWX4uNZeRMZPtXGFPrQGp2n7WHwfsvBOqJq2lr5dldDcoUeteo/so+DfDmqfC+/vdat42Ece8sw9xVP9syaOz8KaNYyHdcrEAVJ9jTvgMZLf9nvXnD7SID/AEoDUi1f9orwHpMlzpMGgK9uoMfnYHX86+ffD/xA0fQviE+qNY/aLF5N3kgDAFcFqMhmvrgFmC+YSR61DDA7OPLjdvXYMmmGp9z+MLjw94++BF/rVhpq2Uiw5Hyj1FfHvgbxNa+EfEkd7dW/2u3ifJjAFfUvg+GQ/st36lJAfI7qc9RXxxeQy280m9Hiy38S4zSDU/Sr4a/FnSdY+D95qsGliKGOPcYQBg9OOtfI3xj+OWieONLuNPsdD+xyrkGTA9frXr/wEie9/Z41qOJTJKIT8vftXxvqllPa31w0kUi/Mcq6kDrTDU+7/wBmvUNMsf2eBc6ov+jLFkqfrS/Dfx54M+MlxfeHE0WOAkMqTBR+fWuV+Gz+T+ytcZ6m3JA/GvP/ANjFf+LkSABurcdulAanb/AX4b6PpfxF8Q21/bJPb2zscMOwxXqPhvVvBPjfQNWum0WOOPTc/wAI7Ef41y3wy+X4meLwRnh+PTgVj/CHL+A/GZPJAfp9RRoGpF8VtN0D4hfBubWtOsFtWt1x8oHrVj4J+CdH8F/Ct/E02mjVbrZu2KoODWRob+Z+zVqoGchG6fWvOvhj8fvEfw10lbO402e+0qTqCuRj8/ajQNTq/HXx28Pa1o11ZXPhtrO4IKpIUAx+tfK87Ce7l25KyHKj0r7y0mx8K/H7wLqOpnRDpt1DGXDMoH+etfHvhvwXL4h8fQaNbDerT7BtHbNGgan0R+y74Zj8A+DtU8Y6igQJHvhZhz26V8+eJ9Yu/i18TXO4lrq42gN3r6H/AGoPEsHw78D6d4N01gjeUFmK/jXzR8J76Kz8fabcXEoC+evzGkI+xPEF54Q/Zy8KWFld6ZHe380Y38DrXVX3iDQPFHwA1vUNDgEPnQEvGMcdK8c/bW0S7vLzS7y3SSS2dQwdBnPBroPhToF7on7Nesy3UMkSyW5Zd4x3FMDlP2Y/gpZ+IILvxDqkAnt7cl1jA5b2r1XTfGmla9rj6DN4Pkt7Bz5QmZBj69ag/Zn8RyWvwd1QWCLNfQoxEY6k5rx+4/ag8W2Wtz28ejyLPGxUq0fv9aQHpfh/4E6Z4P8AjfHBLbrLYTPuEbD2r07Q5PBXijXtZ0A6TEq6fnJ2jsAf615F8HPihrvj74r2j63byQzKcDeMdjXS/DltvxX8ZjPUtuP4LQPUh8Qf8Ix8RvA2v2lhpawT2KFVZVGeCP8AGuT+C/w50DwH8P77xjr8C3OwFkVh71c+DDbdL8bnt83X6rWpr0EniT9l+6gsV8yWOMh1j5PWmGpP8P8Axx4K+P8AJdaKdJSynIKxvgV8gfFzwX/whPjK/wBLAyEkKggdq9i/Yx8N6k3xGW4+zypBA3zs64HSsP8AaMiPib40XVpYoHkefadvNPQNTpP2OfAcX9oTeJr9ALWyHmBnHHb/ABrzf4/+PG8ffEK9mjkLwLJhPQfSvffibcJ8C/gnb6Na/u7+9h2yFeDzz/SvjRpmml3k8sc0aBqfb/wZ8E6P4P8Ag6viptMGq3XlbwiAFs5rgPiF8dPD/iLw/dWN14baxusEKzIAf51znwp/aC8SfDHQorSbTJr7SHGAgXIx+de2W9n4U+P3w91LUl0dtKvooyxdlA5o0DU8s/Y/8K6Z4k1rWjf26zxKrMiuM44FeXahpkEPxZa1RMw/atuzHbNe3fsY2/2bxZr1qCGYFlUZ9hXmuoeFtUuPji8K2MjN9rzkL2zRoGp7x8Rvh3o1n8TPClvDaxxxzFS644PBr1Hd4Oj8bt4RbSYmcgKzBR6ZrlvisrW/xZ8HRuuyRGUHP0NVbyX/AIyaz0XPbv8ALSAvSDwj4qvNc8JWelRxvbqyq4UZzgGvijUvg/4jutRuXsNJmmhEjYMYGOv1r6t+Grf8Xp8RDurtx+AryO4/aU1j4e61e6dBbrIiyNjP1+tIDypvgr4xA/5Adx6cAZ/nWz8Ovh7qekeOdMh1vTZYI5JAMSAV6Af20NdJB+xJuPt/9eqOgfGvUfih8RtF+1QBFjlH3fxoA+hPi94v8D/BW9s/+JKlxeTqucKOD+ftXC/tBeEtA8ffC218Z6XaraS+XvZQPrxXM/txN/xVumgZ3LjIH0NdZqWf+GTYtx5+z5x+NAHxa2ImUH5VHU9qFXevXIJzkV3nwk+GM/xZ8UjSLV/KZ3xluldd8dv2dr74KQ2s1zcrIJucjPv7e1ADf2UfD9lr3xWsba+jWaB3+ZWHFfSHxL+KPgX4V+MW0aPQ4psvskYKP8a8A/Y93N8YNOAIbc39DVr9qfwrqlv8X7mT7PM6Sy/u2VSQaAO2/aI8BaLa6PpXjbRIFWxnAldQOnJ4NHxc8E6Z40+CmneItEgUPFAGlKD61v8Axhjbw5+zPYWF8u27kgGEbr3rK/Y91dPGnhW/8H6gwMMibEVj9KAOa/ZU+F9pcw32u6vCHtLcFhuHHQVU8B/H3T/hv451u8jtswSSMqonTtXsvxds4fgH8GbjSoWCXt0hHHB6/wD1q+Dbi6eaRpWOWfJb/Gm9hPoe5eDvjfBoPxQufEDwM0dw5YLnkZ49a9FuNY/tm5a9UY807gtfItrNIbiEbG5YZavqTQRu0+E9PkFezgj5/NdzRLbue5opOtLXsnyoUUUUAgpzfdNNpzfdNBpHY+sf2DP+P3Wv93/CvFv2iv8Akp2q/wDXU17T+wZ/x+61/u/4V4t+0V/yU7Vf+uprip/x2enU/wB3R5r/AI0Uf40V2LY8jqwooopgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFIaQvQX+dWNP0641SbyoIy7+gplnbNd3EcSjJdsZr0e30C4+H3kajKAyuM4rhxOIVNWjuevg8G6rvPYf4Lh0/w7b3EesRCOVlwN9cbr2rlrqaO2kYWxOAAeKseMvFX/CTXXmhBCBXNFgxyAcetc+HoNy56mrO3GYyMY+xpaIVmLd8UlFFev1PndbiUtFFABRRRQIKKKKACiiigAooooAKKKKACiiijqJ7CN9018+/FT/kaJfrX0E33TXz78VP+Rom+teXjPhPby34ziH+8PpSU6T7w+lNrwIn2XQKKKKBBRRRQAUUUUAFFFLSdwFXGOelTLYTsSRGzLjjim2/+tTHrXunh+zgOlRF4lY7R2rso01M5K9f2SPLvAt/rnh/xBaXOkQTfaVYEBeM19Y2Pxt8Q315Zw33hSdtTBUfaCo6+vWvOtPaPS7mO4jt0Locg4r0mH41XC2yhrOMzr92QD/69dX1W55qzC2ho/tn65JqHgHSLeYf6c0fzRDt1r4YbT5+vkOv4V9R+KfE914uuhPfYm29A/aud+wWoXiFAe/FH1Qf9os+fhYT/APPFvyNegfCDx9efCzxFDqkNtJIQckY6V6F9ht+0Sn8KT+z7Vs5hT8qPqgf2izzH4teMLr4jeKrnVpLaSMzNkjFcjaxXMMgcRMCpyODXvn9n2ox+5T24o/s+2brEufpT+qC/tFm34B/agOm+F4tH13SXvoYl2oWGcfrVvV/2triTR73SbDSJLWylQqqqB/jXN/2fbswAhU+gxU9xoAswrT2mwkZG4cGj6mH9os5P4W/GK98Aabq8CWsha/BJ/HH+FcZ4R8SXnhfxlHrsdvJuWTzCuOvNer/YbQHIijOfakOn2mBmJOnPFH1ND/tFnIfEv4q6l468bx+II4ZYJkYEL64FesaZ+1hHeaTBHrOiNd3kSAJIyg8j8a5b+zrVTjyI8j2pf7PtiMiBM/Tmj6og/tFl74nftS6j4+8GyaCdNe3i27Bgdvzqn8Gv2jrnwLoZ0XVdOkvLBuNpGR/Omf2bbdfs6HPqKsW+gpdSOsFsJGY8ACj6oH9os7GP9r2Lw/ck6NobW6Hg7QPz6155ov7Ql9pvi/VdWNjKDe5z+IHvV2bSYYZijwBXHUEUz+z7X/nivPtR9UD+0WYHg/44ah4Y0/WbRLGUi+3Av9ce9R6b8a76w+H934ZWykKzAjfj3z610f8AZ1rx+5TH0o/s62A4hj9TxS+qB/aLK3wr/aM1bwPpkmmahaSX1gwx5bDPFdrfftRaNZ2brp3hbZPICC2wcfrXJnT7Xj9wox6CgafbHkxIT64o+qB/aTOX8A/GG+8F+LrnWksX3zknaB0zXGal4hvbzxa2t/Z5N5m8zbXrY0+1/wCeMf5UfYLXk+SgOPSj6oL+0Wcb8Yvite/FaLToZLKSD7KoXcR16/41S+DvxN1b4U6213axPJE5+ZfWu/8AsFttwY1xj0pfsNscfuUGOmBR9UH/AGiyx48/aGs/F2i3NufD/lXMy48zaP8AGsL4I/HbUPhiktrJZvcWM3DRH0/OtT+z7XtCh9cikOn2isP9HQCn9UQv7RZ3D/taWWjzx3Gi6AbSUnLkKOf1rwv4y/EO9+K/iCTUZrdon64xXe/2fbcgwIR24pDp9qcfulA+lH1Qf9os8u+G3jjVvhrrkepaekgdeSuOteteNv2iovGmiT2914fIvJlwZ9oyT69ag+wWw/5YR4+lNGnW4yfJUjGOnFL6oH9ospfCH40W/gOze01DRftkDdio4/Wu+1b9qq20mxnh8O6IbOeZSNyqB/WuPWxte0KZ9MUq6dbN/wAsEJ+lH1QP7RZS/Z/t7bXviV/bHiadYIRJ5uJQetVP2nPiBcePvGEkdoryafA22Ir0xit6CJLfcY0EWD/DSNZW88mRArn1xR9UD+0WfPX9m3an/UyflXs3wY+OWtfCxPsj28l1YOMGI84Fbv2K25zbr+VIbG13f6iPcPaj6oH9onb6r+1DpDWsrWHhry7yRTlwo6/nXlXw3+NmoeBvEl/qqWMhN4xJXHTI+tb32S3/AOeKj8KX7Ha8/uVx24o+qB/aLPKr7xFeXPjZ9ea2kYmXzduK7D40fGC9+Ky2JlspI2tVAHHpn3rpPsdtwPKTPfilNnbNj9yhH0o+qB/aLLnwx/aQX4f6Da2y6B5lzH/y12j/ABrd8XftZDxZaTJd+HzLI67RIVHH61yws7ZScQqR70n2O1/55KD29Kf1RB/aLOF+GXj1fA/iCfUbjSDdxzSbhGR939a96j/bOkt9N/s9dAb7L08vAwB+def/AGO1Bz5CA+woW1tmPECkfSj6oH9os4f4weO1+ImoCe00trVRyQBXmv8AZlz/AM8JK+gfsdt3gTP0p32G33Y+zL+VL6oH9os8Cs7G5t7iKYQSHy2Dbcda9i+IHxy1Hxr4PttBispbe3hjCn3/AFra+w23B8hQc88UfZLbj9wnT0p/VBf2izyLwR4g1HwX4htNUt4JS0Mgaus+NvxPvPi5qMNxLZyRypwWx14rsfsdt2hT8qDZ2veBOncUfVB/2izhfhF8Udb+FWqNNa28jW7cSR9mFdR8WPjNb+PNDlt7fQja3LjPmBRnOfrWj9jtcY+zqB9KUWlr1MKEY7il9UD+0WaXwf8Ai54v8LeFUs7rRJdV0oL8nGRj86+jvhn8Qr7V/BGt3N3pv9k2XlNtjYAZ6e9eM+E/iVJ4V06KyS1jlgQYAIqTxF8WtR1uy+wxL9mtm4ZV4BFH1QP7RZ4V4d+IWsfD3x9Pq2nxuytMSyjvzXucn7V1htF7J4e3amRkybR1/OuAaxtpHJMCcnJOKX7Db8/uVP1FH1UP7RZ538VviNrfxS1yXULyKVE/hT0rpfBfxmv/AAn4AvvDwspGFzHt3fl7+1b/ANht9wxChz14oXT7cAnyVPPTFH1QP7RZ4NNY3Mlw0n2eRd5yeK774V+MIfAl81xe6U18h7Mv/wBeu7bT4F+byV59qT7DbshxAmCOwo+qB/aLO2tf2ult9NfTk8PlbQjHlYGCPzrxv4tfEC2+IHk/YdENlt+9hR7+9dc1nbBivlKTn05oexgOcwocdOKPqgf2izC+C/xw1r4Wwz24tHuLKUbWixwf19q0Pip8YrLx5pMlpa6B9llYfM4Udfzq99htmwVgTPsKFsISSBApPfij6oH9os5/Q/jRqGjfDabwmLCTY6bd3+TWJ8HviBffDTxF/aSWsjHJru/sNupAEKnnrikNjbxrgwrge1H1QP7RZneHPjxqGg+I9W1NLKTdfE5/HHv7VX8IfGzUfDei6zpwsZNl+WJ49ce/tWz9jtdw/dKR6YoFlb94lzT+qIX9pM5zS/jLqGnfD+58Niwk8uXIzj3z612Hw7+Pthouhw6fq3h37SF6EqP8ao/Y4D/yxQn6Uv2O35Pkr9MUfVEH9os3vGX7T0knh2fTPD+ktYxTLtbaAP61V/Zej0fQdSvvEWt3CJcQ5kjSQHJPHt71lGztuCYE/KpVVFj2qoUdwnQ0fVEH9pM8z+MXiq+8f+LLvUHSRkLnb9K4i3s7y1uFljhkDKchsdK9+OnW7f8ALFcZz0pBptvn/UqRnpip+qoP7RN7wj+1NLD4dttO1/Rm1AxJtDMAcY/Gl8XftXXfiLwhf6Db6XJb28ybUUAAAfnWB/ZsHH7lB64FB02Bs5iX8qPqqD+0TifhT8YPEHwy1EywQySW8h+ePsRmvbo/2otDuENy/hgfa2+Z22Dk/nXDLpcHQwqR9KX+zoCP9Uo+go+qoP7RI7X9oS7tfHn/AAkEOmPDHuyqKMcY+tQ6D+0DfaP4o1fVvsMn+nkgj6ge/tVr+zYOf3K4xxxS/wBmwNn9yntxR9VQf2izG8G/Gi78NWGt262UjG/BJP1x7+1S/CT4/av8PXntLi0kutOmbBhIyB3rTOn2+G/cpkjHSgafb/8APFM/Sj6qiv7QZ2Wrftapp2nyxaBobWM0ylWkVQOfzrif2fVtde+Ira54nuPJCOZWMn8Z/Kpf7PtsYMKEfSpIYY7fJjjVT7dMUvqof2gzC/aQ+IU/xC8ZzJbKzafA22Pg4NeOx2M+4EROCOMYr317SCRiTCmSc5xTvsdvyPJTHbjmj6qH9oMn+HPx203QfDdrpWp6D9pEIxuKj/Grfjz9pSS+8N3Gj+H9KawSYYJUY/rWZ9ht+P3Sn8KT7DbdfIjz24o+qh/aDPOvhl8Qta+G/igatapKzM2ZFx1r33U/2trSYJe2/h9l1DjdLtHJ/OuH+xW56wx/lS/Y7fIxEoX0xT+qh/aDI/GX7R2p+KvGOmay1lKHtXDEY9j/AI1BP+0NqEvxFPicWUq/7GPb61b+w22D+5TPY4pk1nbsrHyE6elL6tyq4LHuTsYvh/4/zaJ411LW/IZXuySVPbj6+1eUeItUPiDWJ7txtMrl6k8UKo1iZV+XDdB0rHDN90/e9a8+UeV2PYp1HJXY4RhRjfnit7wV4jPhPxBbaki7mjYH8q5/IWkyMNyeaks9N+Mnxhl+KusJeTQNGY8Y5HpWhN8eZ5vhavhQ2zbPL27s+/1ryBQFxgZPfNHRsgcZ4pAejfBX4pSfCrxVFqscLSbTv25FdZ8fv2kZvjVZ20ElqYPJ6En6+/vXhuAM4GKXcRnn6UAe9/sdyBPi5YSE5Kv/AEr3P4w/tLaT4d8cXVjqWlLdPbthJCB/jXx18PPH138Pdej1SzH75DkY9ar+OPFl1441261O6YiaY5+lAHcfHD46aj8WrpISrQ2UXEUY6V3P7I3hHX4/G1lqcVvJbWCtuklbgYr5yjykqPkkqcivZNJ/ae1zw/4ZXSNPjECbdpkXrQB1X7ZvxKPi7xw1hFL5lvbtgYPFfNxb5jnrmrWtapPrl9NdXLs80jbixNU8HdQ9hPoT2p/fw/7w/nX1Fof/ACD4P9wV8t2vNxF/vD+dfUmhD/iWwf7gr2sEfP5ruaA7UtJ6UteyfKhRRRQCCnN9002nN900Gkdj6x/YM/4/da/3f8K8W/aK/wCSnar/ANdTXtP7Bn/H7rX+7/hXi37RX/JTtV/66muKn/HZ6dT/AHdHmv8AjRR/jRXYtjyOrCiiimAUUUUAFFFFABRRRQAUUUUAFFFFABRRSfpzik9Nx7i1JDbyXGdilwp5wKl0/T7jU7hIbeMySN0Ar0HwfDZ+E/PTWIRG7DjfXDiMSqekdz1sHgnV9+eiQ/SfC+lR+HTfNOq3arvCnqDXJa94uvtXjFvNJmNRtWo/EWrCe+nW1crbseADWExycmufDYZ83tpnbi8bGMfY0dhaKKK9f0Pm91ZhRRRSDZWCiiimAUUUUAFFFFABRRRQAUUUUAFFFFABRRRR1E9hG+6a+ffip/yNE31r6Cb7pr59+Kn/ACNE31ry8b8J7eW/GcTJ94fSm06T7w+lNrwIn2XQKKKKBBRRRQAUUUUAFFFFAdSWH76f71e86B/yB4f90V4Nb/6xP96vevD/APyCIf8AdFejhTx8fsafllgO/tViPTbqRdyQSY9hW14H0eLWvEFrDKwVCwBz3r3XxhrFn4CaK3ttHM8YHLKowf1r1D54+aZIZI2w6Pn/AGhinNDIsecMF+ldv498TWHiCSGS2s/szj7ygYrstW0Gx/4VVb3iwhZ2TJbHNAWPF4beSZiVVnx/dFCWs02QEyR1ABzXrvwO0m11CO/W5hWRQDjcPpXReANP0y2j1i7uYFkFuSV3D6UXCx4H/Z1wmP3LAt04NMntpbcgSqy/7wr6hg1TQ77wu2sPYx/ueQMVx3xW0+w1Dwja6raW6xbhk7R9aLhY4D4VeFR4k8RxJKuYUbLVd+MepW02vmytUVIrc7QV7113wDtRb6bqV4Fw4QkN3HSvJfFN095rE7udx3mi4WN/4c/Dr/hNbx2e4S3gj67q9L8ZfBvRtF8G3F7bSrNNGv3lrw/StevNLLrbStGrn5tp617vZXkt18H7mSRizNHzk0XKsfP9nYSX19Haou52bbmvebP4Z+GvDGm2/wDbM0f2mZQdp6g15r8KrFLrxlAkoBy+cGvXfF3h248QePFWbIsbcgknpwKLhY4j4pfCu18P6VHqemuGt2G4gDtXFfD7VY9L1+2Mih4t4BBr6M8ZahZeIPAN7HAB5cMW0Y6DpXypbv8AZdRBRs7XouFj0n40eE4tMul1C2QLDMNwwK8txuz7V9AfEJRq3wmsL6RcyCPgnr1NfP2eDii4WFwPWjA+tJTo18yRUGSWOKLhYfDbyzNiFWf/AHRUk2n3Nuu54ZEB7sOK+gvCfhKz8H+FI9Slszd3Ei5VcZrW0u1s/H1lNa3mli0kwdrFfagmyPmRLSS4U+WjN67RUXLMPXpjHNep+FpLTwp4ul0vUo1a2L7dx9K6+6+C8D+IlvlCjSt+7cOmMUXCx4DNbywx5kRkX3FOgsZ7nHkxPKP9kV6V8Rrix1bxBBpGlIoRWCFh3r0iPw7afD7QbZk05r27dcnCii4HzbNZzQ/61HT/AHhUar6Y/GvpG50S08feHLuRtNNldRrkEgA15/8ACv4fR63rtw10M21sx3r7CgLHmiaXcyLuELsP901F5MiyGMowc/wkV7trXxO0LQbySxttNWSGI7S2B/jXITappniXxpaNbQLHE8gylFxnm7xvE21gyH/aFL5DiPequV7tjivVvjtotrpeqWyW8SxI2OF+lbf9g2f/AAqA3vlIZvLyGxznNFxWPL/BPheTxFrUFqyFY3P3yK0fiJ4FbwrrjW1uDKmcEqD6V3Hwd8dWcOoWunGyDT7seYBzXS/FT4jWWi69Pbz2IkfuSBnp9aAsj5wMbRsRgq2fumu5+G9hp32qRtWRljxldy8VzWpatFd6416kBEed2zivdPhn4i0vxdbyWf8AZ6oyR43EdaLhY8H8RRW/9sTpalTApOOtULe1kuP9XHvx2UGt3xhbpD4luYYxtG/aBXseg6XpPw+8ExarfW4muZEzgigLHgM1nNb8vEyj/aFRwwvNkxoxI9BX0Tod1onxZ0m6gFotvcKp28CsH4N+F7RvE17Z30aukbEDcKLgeNx6dcyKdsUpI9qrspTKurBx2NfR15478PaR4oOkR2Mbxl9hIFcL8dPDFpoOpQ3FtGEjn+bao6UXCx5VgetB2ryTjFJR7HOPai5VjS0PSZNUvo4ljaSN2ALAdK674kfDceE/s5tlaTzBkgCrvwv8cWWiutpLZiWSRgAzCvWviv8AECx0FrT7RYecXXIzjj9aLisfLLRvHIVdQuOoYHNdZ8PLWy/tqA6nCTbMeWK8VQ8UeILbWtZa7htzDHnO1cYr1v4Q+KdM1qa30ubTl3YxuIHP60CseXeP4LGPXrgacQ1vu42jpWAunXTKCsUhU+grtvipYxWPjeaCFfLi8zgV6/DPpHhXwjps8lpHLJIAGOKAsfNUenzyZRImdl/2TUUkBjbDKA/cHNfWOsXWgaTqFjGlmhN4BnjpmuE134ZW+q/EhYLVAtoz5PHA4ouFjw6PTbiZN6xPj2U4qKSJ42w4KN23DFfTOralbeE7hdPs9F+0xINruFH59a5f4weDbS48ORa3bQfZnI3MmOaLgeHrDJIwCBpH/wBkUk1vJbnbJuDHsRXo3wXOmLr4OplFiLfx9KsfHJdJ/tZf7LePZ32U7jseW7fUYo2ijbtY85op3HYnsrcTzRo3CkjNfRL/AA38J6HpFlcalIsfnRgktXz3psyx3URb5VBBLN9a+k9WuvCvi/Q9OtrzVoUZEGevH6UrhY848fab4Os9Kb+yZ45Jz06/4VjfCn4dr4yvna5KxWycmQ5xiui+IHwjj0vSRqGk3SXdpjO5Mmuk+F8b2XwxvZ4FzOUxx1zkUrisSw/DLwj4guJ9O025R7yIYyoPJ/KvHfEXge60nxINMCkyeZsHFe4/C/T7fwUovr9sXty2Qp61Z8Wafb3fxSsJzF8sr7sfgaLisc9Z/DLw14ZsbUa1Ki3kwA2Hr/Kuc+LPwstvDWnx6np0g8hhuxg9K7TxVoM3ibx8WnJSwtm3Fm6DArf8YapZ+J/AdykQDRwrtU+mKq4zyb4W/DG18RWMmq6lIkVnGMnINdpJ8LfDPinTbltIkUzwqT8o6/pU1rFNa/ClYbJfnkXadvU81v8Aw7tbXwXDaWMrBr66X5vUdaLgfLWp6RcabfzWroWdXK4A5qGHTbm4JVLdyV6/LXvUehWlx8Yfs8yK0bSZKkcGtjxj4w8P+CPEP2GKxjdmYK2BSuI+aZbd4X2SjY3oQaILWa4bEMTuf9kV7h8afDtg2m22qWcSxLMuSqitXwH4fsdB8Crq6WgvLkx524yc5ouM8Am026t13SxyKP8AaFVevUZr2jxB8SLK+sLi3udL8hyMAsvSvGpMNKWHQ0XFYN1G6kopD5UO3UbqbRQPlQ7dRuptFAcqF3UbqSigOUVqSiigYUUUUDCiiigAooooAKKKKACkk/1bUtNl/wBWaUthx+JHhvir/kOT/wC9WKzYetrxUP8Aidz/AO9WLjMleBU+I+uo/AhDSUGis3ubBRRRSAKKKKACpaiqRTlaAFppT0p1FAEZHNLj5qX0opPYUug+1U/aIv8AeH86+pdC/wCQbB/uCvl63/10X+9/WvqHQv8AkGwf7gr2sCfPZruX6Wkpa9lnywUUUUwQU5vumm05vumg0jsfWP7Bn/H7rX+7/hXi37RX/JTtV/66mvaf2DP+P3Wv93/CvFv2iv8Akp2q/wDXU1xU/wCOz06n+7o81/xoo/xorsWx5HVhRRRTAKKKKACiiigAooooAKKKKACiiikxhU1rbtcTRoBnJ60lvbPcMNqMxz/CK9K0Pw/pMfh03UsqpeKuQp65rixOJjRjZatnr4LBOvLmlokMi8OSeA4YNVLrKMbttcv408WHxPceY0WwLUeteKrzUojbyyM0anAya59j8xPrXLh8PKX7yfU7sZjIU/3VIGO7nOaTrRmivZtZWPm73d2LRRRSJCiiigAooooAKKKKACiiigAooooAKKKKACiiigAoooo6iewjfdNfPvxU/wCRom+tfQTfdNfPvxU/5Gib615eN+E9vLfjOJk+8PpTadJ94fSm14ET7LoFFFFAgooooAKKKKACiilFAElv/rE/3q968P8AGjxZ/uivBof9an+9XvHh/nSIvpXo4U8fMPhTOh0Ga6iv0ks93nocrtr2fRvixd3EsNjq2lSTnIUsV/8Ar1494c1o+HtRiu1j80KeV9a9TT436ZxI2kjzlHXA/wAa9Q8El+OHhWw02Kzv7JBAZhu2AfWtS8tpL74O24hUyssfO2vL/HnxIuvGkyFx5cKcKvpW54B+MD+GdMOnXkP2iDoF9qBnX/AfSbuC0v5ZYGVCpIJGKm8LfNoPiI4z97A/EVif8NAC2WWG0sjBCylflx/jXIaf8SpNN0/ULaONybon0oA9F0v/AJJXe5+6Bx+Ypniz5vg3aHkgp/U153afEqS18MzaSImPmDnJFGofEyTUPCMGjGL5UXFAHbfs930c0Oo2LN8zqQinv0rzDxhYPYa9cI8bKN561a+H/is+E9eiut2yPdhvpW38WNW0rWtS+26fOkjEbivvQBwdpZzXcwSBGd93RRX0Npen3C/CKaIwv5hi6YrxjwX4sj8L6h9qltvtHOQvFejx/tGCOzFsNNcIRgqMdPzoA434es+g+M7WS7UxgyY3MMV7p8Ytak0vRwthC0kt0vMsYrwHxl46i8S30dzb2xtXjOR0rtPD3x5Eemx2+o2f2kxLhS3/AOugDpGjfw78JZmuz5c9wmSrda+f7NDd6hGqDlpeAK7b4hfFa48ZQrbrG1vbLwFHSsj4eyWFvr0M+oSBYVbIzQB618ULhdE+G9lprHnyhgfnXz6ecgcGu8+LHjQeKNWaK3fNrF8q+lcGc8n8qAFqW1bbdQsTgBxmoqN23mgD6l1zxNd2PgLTrjSoftSrGNwH4157a/FjxHdO621iytjB2riue8F/Fyfw7afY7mH7TbYxhq6Sf42abHAyW+lqrMMZAH+NAHEWen6l408XKhik+0M/ze1fSdvqOnWekR+HJLwNeumzJPINfPfhf4of2DcT3aWuZ5s7WPb9a5+48YX1xr39pmZvM37hzQBv33h+58I+O1S63FfMyJG7817P8UPGGpaPp1lPYQme3ZB86jNeLeLviSPF1nAksBW4iGBKvWtPwx8apdPs0sb+1a7hQYG/H+NAF+P4q+I7u1nSG0kAx82Bit/4F6gNQj1a3kZVuZ1I5654rndc+M1ncWE0FlpiweYMFsD/ABrgPDvjC78M6oby23ht24gEc0AWvFfhu9sdeuIpIHMnmHjB5qbwXp8+m+KrH7TGYcuPvV30fx2sbmLfd6UHuCPvEDP868/8TeNn1zWlvoYjb+W25VXFAHov7QGm3N5qVjJDG0icEMo9q3bqxnsPgn5c8ZjkWLOG+tc3Z/HuGTSIor/T/PniGFZsf41leLPjfP4i0SbTBbNFE67R0/xoA5/4VuF8dWbNhT5ldH8fNNuJfFksyws6MOoFebaXqsml6lHdwkiSNt3PWvUJfjst9axx3mnGZwuN/H+NAHkjQurlduHHG3Fe0fs82ksGp3DGMooGSSPavMZPEUb6+NRFsfL37jHxXpGmfH630gf6NpnlnGCQB/jQBxXjizmh8XTyvGwHm56e9er/ABIsJta+HNhPaR+cixgnb261wviT4vW3iG1mi/szbJJz5nGf503wb8Yp/D9n9guoWubbHCtjGKAOi/Z80m5sbu7u7mMxwgEndxxXQ/DO6S78d6xIm0ozsAB9BXG698cPtWmNbabZ/ZTINrFcD+tcr4F+I03g+8uLgoZJJjk/lQBa1jj4iNjn9/8A1rtv2jFH/Ev5J+QfyNeS3niNrrXDqYTD7txX8a2/HnxCk8cfZ/MTYIV20AcdSdO+KWms3y+/agDX8MSBdatdxxiQc163+0FY3Ey6bNHC0ibB8wGfWvE4J2tZFkThgcivVrH45uumRWt9Z/a/LXaC2D/WgDySS1lhfY8eD1wa9Q+BdjOni22d4iE7cVyGueJE1fWBfJbmNQc7Biu80j45W+hpE0OmYkRcBlxnP50AZ3xktJf+E9mkETFfM6gcV3Hjf934J0YkcYXr9TXL6p8b7fWVkabS8zH+IgZz+dYPiL4pTa1ptva+UwWM8DigD07xo2dY8OkdF2/1rr4tQgtfH0QdljLj5Q30rwLWPijLq1xYT+S4e2A44qDxB8TLrWtYh1CLdDLCeMnrQB6X4++ImtaL4hmgFg3l5+UgZ3Vw/jL4ga9rWkNb3du8Vo46kcVu2nxzgmt4lv8ATfPmC8vxn+dcz46+I0fiqzWC3tBbovHTmgDF8EeEr3xhqHkWkpjYtgYqbx/4LvvCF55N3L5jN/epvgLxo3gfUBeInmEHIWpfiF4+fxxfC4aPyzQBx479MdsUtJ/CB1paAH28MkzlYwcngVoSaXrFvIEeGYHptYVX0zUG0+5imKMzIQdvFeww/HDS2t18/SQ0qjrgdfzoA6bwPHc2Pwxuf7VDqhj+VZO1WfgRqFrf6Hd2bfvCufk7nmvKfGvxevfElr9lgRra2x9xSOlYfgfx5deC75ZoPMZOrLkUCPWnuLzxN8SLe3FvJHZW8npxVrxf4otrP4o2MQfKQOFJrAufj5AsDyWthsumGN/HX868k1TxDdapqhv5HPml9xJoFc+mPjNrTaZpu2wtWaW7XBkQdP8AOK51lfw18J5Bdr5cky8K3WuZ0f47LHp0VrqNp9qaEYDH/wDXXL/ED4pXPjVBbhGt7VOAoxTGe8fC/VLK++H4dlE7wR5Kj1zXIeBrnUPE3j6XU7qNo7aBiVDDgCvMvAPxNuvBbMgDzW7cFOMV1up/HZP7PljsrH7PLICCw/8A10DNrQtTTVfjN5isGHmYBH0rj/jIw/4WBLg5AkBzXKeFvGUnhvxAupjMkobdyaZ4x8WN4m19r9k2kndikB678Vs/8K707k7vLH9a5PwP8QNZ8J6aI3tpJ7PGApHBrG8TfEyTxDotvYGPaIhtBrZ8L/Fy10vTFs7zTRMFGA2B/jQB6TbxaX8SPDN3cS6ctnPGhJLLjmvmu8hWG5nQcqhIFepeIPjiZ9Ney0yz+zLIMMR/+uvKpZPMkZjyx60AMooooEFFFFABRRRQAUUUUDCiiigAooooAKKKKACiiigAooooAKbJ/qzTqbJ91vpSlsOPxI8O8Vf8hyf/AHqxf+WlbPir/kOzfWsb/lpXg1PiPrqPwIa3WkpW60lZPc2CiiikAUUUUAFPVgFplFAEm4etG4etR0UAPz0ozTW60dzSlsJ9CxbsPOi5/i/rX1DoLD+zYOf4BXy1a/6+L/e/rX1FoP8Ax4Rf7o/lXtYI+fzXc1OlFDfeor2WfKhRRRTBBTm+6abTm+6aDSOx9Y/sGf8AH7rX+7/hXi37RX/JTtV/66mvaf2DP+P3Wv8Ad/wrxb9or/kp2q/9dTXFT/js9Op/u6PNf8aKP8aK7FseR1YUUUUwCiiigAooooAKKKKACiiil1DyDBq1YadcajcLDbxmSRjgAVFb25nnjQL8zHFekx+F5fBNpBqwZWbG7bXDicR7L3Y6s9jB4JVnzT0S3E8Ki38HSOmr24Bdfl8wVyPiTVxdahObU4hLcBelS+LfGUnieVXaPYFrmRkd81zYbD+0l7SodeKxSpx9jS2HNncSTkmiiivXPnbvcKSlophe4UUUUCCiiigAooooAKKKKACiiigAooooAKKKKACiiigAoooo6iewjfdNfPvxU/5Gib619BN9018+/FT/AJGib615eN+E9vLfjOJk+8PpTadJ94fSm14ET7LoFFFFAgooooAKKKKAClFJSigCSLmRPrXvOgcaRF9K8Hh/1if71e8aD/yCYvpXo4U8fMfgRo/wg9aOxPShR05/Crkej3kyb0gZl+leoeEU9vJyePSk2/KM9qlmtZISBNGVPuKGtZRGHMZEfY4oAiUDnuKXGGBB4xUsVrLOCY0JA9BSw2U87MI4mZhxwKAIVA6A8+tC9zmrkmjXi7Qbdtx6fKagvLKa1GJIyn1FADI0Z2CgbmPpVi6025tFQ3EToGHVxXXfB/wqPEviRBIMwxHcxq78YtZt7rXJLG1VRDb8EigDzjqTt6rxinbhkDBzjvXcfDXwCni+5d5rhYYV6sTXpPjD4S6JpvhG4vbOZZpY1++KAPn3ngDt1obO3IB/CrdlYyahex20PLs+2vd4Ph54a8K6bb/21Iv2icDKkdDQB8/bflH6ZpPukDJODmvYvit8LbbRtJi1PTRm1Zd3A7VT+FHwwt9dsjqmqNss4xk5oA8qbPPBB96Recdq+i4fh34X8WWdzFpLhp4wRkCvHbHwPc3fipdGIwwl25oA5jpnJ+nFNB3DP519FyfD/wAH+HZLfTtRmQ3b4H4/lXA/Fj4Zp4RkW5tW32cnK4FAHnlrpt1fIxtoWk5x8oqBo3TIYFWB/ir0H4OaxFZ68lncIrQynb81QfFvww3h/wARyCMYt5DlTQBwf8QHtSr8rZHHtSfTnPWloAQfLgUY9DignbU9vZTXn+riZvoM0AV8Zzkkj0NAx05xVi4s57XiWJl+oxSJZzNGWSNtvrigCEL8uDz703aG+vvUiozttUEt6U6e3kt1/eoR9RQBGc5Jz+FHHGTk1YttOuLpcwxMw9hUdxayWzYliZPqKAIto6YxQ2N3PIpyxlvlQEsau/2Le+XvFuzD1waAKP8AFx6UKCMA81IlvK77Ap3+mKSSF4W2uCGoAZ97IzRkZ45PtUj2ssagiM8+1dp8NvALeK9UEMymNcZDYoA4f7vGepoU7gea6PxX4Tm0DVJbZEaRFJ+bFc95bO2xV+bpQAm09QD17UjD1bp1r0fwPb6VZ6NfnU4v3zL+73KfavO58LNJtIK7j+VADKO4Pap7WxuLziGJmHsKLizmtH2yxsp+lAEG33707jPvUlrayzb9iEj2GasLo940e/yG2+uDQBSopzqVYhhtI7U2gBRlm60oUHI79jTemSeldL4H8LTeJNet7V1YQyfxYoA5s57HmjncMc12vxC8Dy+F9XNtBGzRdyBXFtEYZNmMNnj60AIAZFz15wc0dD7V33w8trGG6nk1aIeTjC7lPXFclrXkvqU3kYKbjtAoAzgfem54wSS1Xv7JupOUgYg98GkXS7yRGYQNhevy0AUhRnnrTvKYnbjDdxVpdHvGh3JAzD6GgCn/AAjPB9aB0FLJmHhwQfpT4baW6b90rNj0GaAIwuKMfNn2xT54HgYB0wfpTaAE+7nnHOaFG7j8SKCNwxU9jaS395FDEP3sjbQKAINpChz0NI3ygbjw3TivoC1+H3hnwfo8U+vTKJpFz5bVxXxBt/C8ljG+jSo8n9wUAeabcngdO9P2sMtg9K9b8FeH/CNvocV5rFzEJJBnyzmurh+HvhXxpYzHRp4xOi8IvWgD52A+UDp64p3PocZrqdJ8DXGo+Kv7JGQ/m7GNewjwH4O0C4g0vUJla8Ybfx/KgD5zUAqO/fFLzuI74r0j4tfDH/hD7xZrb/j2lPyn2rd8E/DXTbfQzrOvOFgxkZoA8ZXJyM4OKd2JA78mvftW+F+g+JvDs19oJyYlLHArzLwD4Fm8UeIjZyArEjYagDjfw/SgtggV9GQ+A/BlxqJ0ZJla96DA7/lXkvxP8Dv4J1o26ruiY/KcUAcbuweaFYN0qymn3TBcQtk+1F1ptzaKDNEyj6YoArc98UtKsbSsFjQsenFW20W+Ee827bev3TQBTopWXacEbTSUAFFFFABRRRQAUUUUAFFFFABRRRQAU2T7rfSnU2T7rfSlLYcfiR4b4q/5Ds31rG/5aVs+Kv8AkOzfWsb/AJaV4NT4j66j8CGt1pKVutJWT3NgooopAFFFFABRRRQAUUUUAK3WjuaG60dzUy2E+hJa/wCvi/3v619RaD/x4Rf7o/lXy7a/6+L/AHv619RaB/x4Rf7o/lXt4I+fzXc1G+9RQ33hRXss+VCiiimCCnN9002nN900Gkdj6x/YM/4/da/3f8K8W/aK/wCSnar/ANdTXtP7Bn/H7rX+7/hXi37RX/JTtV/66muKn/HZ6dT/AHdHmv8AjRR/jRXYtjyOrCiiimAUUUUAFFFFABRRRQAL82KmtbOa6ZhFE0m3+7T7DT59SuEhgjMjMccV6D4ahTwLcE6rbYDjjdXBiMSqa5Vuevg8G6r53sWPDWh6QPDck1zKsV2q7sN1BritY8UXd/G9q0u6FflH0o8Va2uoalI9t8kTHAVelYXryODjFc2Gw0n+8q7nZjMZGK9jR2Dvn2xikxS0lexGyVkfPayd2FLRRSJCiiimAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRR1E9hG+6a+ffip/yNE31r6Cb7pr59+Kn/ACNE31ry8b8J7eW/GcTJ94fSm06T7w+lNrwIn2XQKKKKBBRRRQAUUUUAFFFFAdSW3/1if71e9eHh/wASiIf7IrwWD/WJ/vV714f40iH/AHRXo4U8fH7Ha+AtGg1bxFbQTsNjMMivc/GesReBmhtrTRzPbqOXVQf61876FJeJqCNZAmdTkbeteyaH8VNRaSKx1XTXuBkLkrk16h88cD488WWfiKaJ4bE2zKeVIxXbazo9ovwnhuliAlMfXFT/ABx8LWFhZ2V/axLA8gDMgGCOtXb+1k1H4OxeQDKRHyF5oGc/8DdFttUS+W4jDMqkjI9q6L4cWdhaw6zdTW6yiEkgEfSofgPo95Yw6hLNA0abD8zD2FT+El/4p3xHjgDcP1FBZ00OraPfeGW1lrBQYBkLjrXGfFSzsdX8F22sW9qISwzwPrVjSHb/AIVbe4JwBx+dReLP+SOWWSf9X/U0AQ/AC1W30zVLtfv7CP5V5F4kmafVryR+TvNesfs93ySQ31gzANIpAHrwK8w8cWbafr11CV2ncc5oAztN1y600FLaUwhjyAa950+4kufg/cSSZaQx8tXz5Z2k15KqxIXbOcAV9F6Vptz/AMKjlhMTCTy+FxQB5P8ACu3W48aW6snSTNeu+L/D8viTx2onyllb8nPTivJfh40ug+LoJb0GDMvG4Yr3b4va1Jp+kL/ZsJlnuV5kjFAC+LtSs/EXw/vIYADHbpgGsDTYZYfhKEtgTJMu3j1zVbDeHfhTM938k00edp69a6P4R6rb6l4AKuvmtEm4KOuaAJvhxZ2ngtLeynYG+uh847j/ADisrStGiHxclYDgNuH5Vl+CJtQ8TfEKTULuJ47eBifm6DFT6H4lh/4W4QZF2+ZtzmgB7eHf7Q8b3ur6idtrA/y7vTitP4wTW2vfDz7XGuVVODWb8btUvIbmLT9Ogfy5Ww7IOtVfiRMugfC+0sZHxK8fKnrmgDwjw3dNZ61aSJwQ4r2H47wrd6Ppl2RiRkBJ/OvJ/B9g+o6/axohf94OBXqfx8vVgttP04MN0agEenWgDxDd8xJ5yaWkb5enOTS0AWNPs2v72KBBzIwXPpX0lH4Xg+H/AIftGh0z7fcyJknFfPnhSZbfXrV2OFDjOa+jviV4u1XRtLtZdKh+0W5UZKjPagDPuNAg+IHhu5M+mfYbmNTjgD+tcL8LrzTIr+40LVI1O9igZu1NX4peKLiynVLR0Xb87YxWJ8PfDN/4s8S/aFDBN++SQdBQB6NpvwatNE1y41G+YHT4f3i56EVwt8sHj7xslnZQiKDfjgV7nrWo2PijSbrw/bXSm7SPZ15Jrw/wDZy+FviLFHegwssuPnoA9S1ixg8AxW9rYaP9rbaAzYH+NZ3jPwfaeKfBsmqC0+w3QXcVxUvxY8c61oeqK1nb+bbsPldRkYrzzVPib4nv9JmiltWS1K8nGBQBpfB/wXZyWt1q2prvht+dpHWruofGTR4bprVNLX7MDt3Y/wDr1p/DGX+2vAOoW0RH2lkOF7npXi194d1CO/lha2lDbsYK0Adr4POn+IPiRE8UQ+zyP93HtWd8UdLg0/xs8EC7Y93Sn/CO3l0r4gWkVynlMHxtb6Vr/FjQ72bx8zLbuyM4wyjg0AanxD0m1s/AGl3UMK+btBbjr1ro/gr8QLbVL5bJLDY0YxuGOePrVH4uWbWPw50pZAyOqYI/OuU/Z9kQeK8lx8/A/KgDe+IvxRtLTVryzfT9zZKh8Dj9a8fsdSjtdWS7ePcgO7bXS/FTSbs+KLxxA+0yH5scVxcNvJJIQikt6YoA+ifC+sab4z8L37ixWN4kyGx9K8FtbEXuuGBRuV5NuPxr2D4N2NxB4V1ZXR1byyOR9K8x0m1lsPEsUk6NEgl+8Rx1oA9m1STSPhL4ftDLZrcXMig8jmkubPSfiV4MuNRgtVhuI03bQKz/AI+abc6lY6Xc2qtNGIxnbz61b+E1jNo3gXUbi7Vo42iOFf8ACgCl8D/D+n3FnqjX8IkWLJGR9K1tE8faHq3iH+xRpyiJm2BsVR+DcwbT9dYNu+8f5VwXgXI+ISnP/LbFAFb4v+H4PDviaeOFdsbNkAVwtepftADb4qlHv/SvK1+7QA9W2nrg17Z8GfH1t/aVhphsQJM48zFeIthctiu2+DrLH43sWZ+h79KAPXPiz8SLTQdaltpbESt0zj/69fP+o6lFf6010ItsbPnZXo3x50y7m8USSRxO6MchgOK8qNvI0m3a28dqAPob4ZeJNL8XxS2T2CoyJgPj2rx+5tYl8YNBsyhmxiu++AFnOl9dF0ZRggEj2rj7ywnj8aO7RMEE2d2PegD3e61DSPD82lWb2Kk3IClsDjrTL680a18SHQ0sVAm43Y9qwvGv/Id8PjduGR/WjU9zfFy1HXnp+FAGL4T+GVtqfjW9Lx74IWzsx9K6q61mKx1T7FD4fY2qnaX2jn9a0/CF9Fb+LNaiBUS/wr3zgV534l+JXiDSdWntmsScOdrbetAFT44eBbbRzHqNrEI45BnbUXwFuNGhurv+1DGit03/AIVz/jjxlrniCyVNThaBAMLuqL4b+Ab3xxJItlJ5ZTqR3oAd8XJLB/Fkp054zb7v4a4VenXNdF428N3PhbWpLS4cu6nrXPUAFdt8I7WK88ZWUci7vnriG6dK7b4TX0Wm+MbSRmCjeASaAPd/iJ4X0C61ASa1erErfdjJry/4hfCuz0nSRq+jzrNZMM4Ga0Pj9aahda5FNbo81u/K7fStvTY5rH4Q3SagSrmL5Fb60AfPc1xN5YRslN2BivevgjpcnhvSrjWrsmOFUyu7vXM/Cn4Wv4ivP7QvExZKd4LDg1f+KXibULjdpGkWkkdgnyZUcGgDQ+Et1DrnxCu7wYwJCR+VdBH4ejvPGl9rOptst4Hym78K8/8AgHdrpnjDyrg+XluVbrXc/GrUr1b+LTNOgkEUjAMyjrQBtfFueDxB4OtbxVyOMH2zWd420ue88IaPpdoh23ChWx6c1X+JV1H4f8C6faO480KAV/Gut/tq3k+HdvfW6+fPbxcbeoNAFnwXDp/h2zOhQlXkWPD49cVxnw7sUs9a16VFy8ZbH6VD8I2vJptT1fU9yEgsC/4U/wCD+uW914u1SCRwTM5+U9+lAEHgPS49DvLnxDqbbZDJuQNTvj9HHqa6VdkDEjD8uar/ABDuL3U/FsGkWtsyWqychRVf443iwnRrEP8AvU2gr6daAOh8VXmh+B9Dsbt7FZZmQHp7VlapJpfxE8A3eoRWi288UeRgVkfHaZv7B0jqAEH8jT/h23/FsdUbOcpj+VAGf8C/CVlq17dT3SLM8OSEPetTX/iUmj3k9lcaIywglN20Yx+deeeD9e1rw/qElxpyM6Bstsr1vwb4wh8d3hsdR0o72GGkZaAPAvEN7BqWqSyW8XlRE8LWZ7V13xQ0S38PeJLmC2KmINwq9q5FTkA0ALRRRQAUUUUAFFFFABRRRQAUUUUAFNk+630p1Nk+630pS2HH4keG+Kv+Q7N9axv+WlbPir/kOzfWsb/lpXg1PiPrqPwIa3WkpW60lZPc2CiiikAUUUUAFFFFABRRRQArdaO5obrR3NTLYT6Elr/r4v8Ae/rX1Dof/INh/wB0V8vWv+vi/wB7+tfUei/8guD/AHRXt4LY+fzXc02+8KKKK9k+VCiiigApzfdNNpzfdNM0jsfWP7Bn/H7rX+7/AIV4t+0V/wAlO1X/AK6mvaf2DP8Aj91r/d/wrxb9or/kp2q/9dTXFT/js9Op/u6PNf8AGij/ABorsWx5HVhRRRTAKKKKACiik69KA0uFS29uZrhUA4zjNPtbKW6k2JGW+lek+GdI0WLRGkvnWO6C5weua8/FYmNNcq3PZweCdSXO9hq+FW8J6fBq0MoZ1Gdtcr4u8aTeKJEMi4K8VBrfiS7n324n3wA4UZ7Vg/eXJ61hh8PKXvz3OrG4yMV7KjshN2adRgUV6+2h83o3dBRRRQMKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiijqJ7CN9018+/FT/kaJvrX0E33TXz78VP+Rom+teXjfhPby34ziZPvD6U2nSfeH0pteBE+y6BRRRQIKKKKACiiigAooooDqSQ/fX6171oOW0mEf7IrwaD/WJ/vV714f8A+QTbn2FejhTx8w2Om8L6yfD+sQ3WwPtbJBr1mP46abGpc6TumHRgB/jXiTMRJwccU38fevUPn0tDsvH3xBufG0ymQmOIdErb+H/xgfwrpzWVxb/arTptP/6680ycg96TrQOx7JN+0E0MckVtYmGGTgquP8a5LT/iZJY6ffW4hOLrrXD4zyetHFBR29r8S5LXw3LpiwHbJxmm6p8SJdU8Kx6MY8Ii4rixxR6+/WgDpPAfio+Fdbgui22NCM+4rb+LWqaVrWqJfafKrNLy4HauBSMyMFUZPp61PcaXdWih54THvHAPegDb8E+K4/C+oNPJb/aFzgbgK9OT9orELW62OI+m3jH868MZT91hjvShSx3YxzigDrfG3jw+LLyCe3tzatE2flrs/Dvx6+y6SlvqVmbh4htVm5/rXj/pzx2o6DgZH8qAPQPH3xVufGCLCkbQ2yjhBjFU/APxJuvBNwQN8lu3VOMVxeO5OD+lCg7eOfpQB7PrXx6SSzkj060FtJIOWH/668qtddurbVjfCU+fu3ls1nBSMZ+UUiq21+OP50Ae2ad+0FF9jRL6w+0XCDh2wf61wPjvx/ceNrwTSEpEpwI26AVyW0gEjr3puRg9/agR6B8J9S0vRNUe9v54xt+6pzWJ8QPEzeKdcmulk3RFvl54xXOD5ucc+lI+APmGPp3oC4fxH0o3c4p0SNMcKuWPAqe80u5sSPtELR7vukjrQBCkhjZSODnrXpnhH40TaPYpZ30AvYgMDfzXmCn7p707OKBnrmvfGu3vLGW2stMWIyDBYAf41ieH/is/hzSJ7S2tyJZf+WnpXntHtQBvaT4wvtJ1tNSWRml3ZcZ61s+NviAfElzDdRQmCdDncvXNcQeetFAHrvh744GLT0t9TtDdlBhWbGf51T8Y/FqLWdNls7Wx8hJeM4H+NeX5PApOmPbpQB0fg3xreeDb4TW7bkJ+ZK9JHx2s2jMjaXvnI6gD/GvE6KAOhufGFxL4mGrxDy5FfcBXpUv7QEVzap5unrJdhf8AWY/+vXilHbHagD0Pxz8Vp/GmmJaSoyFOnpXI+G/EFx4X1OK8ts70O4qDWV2x2o9aAPXNQ+Ny6pbPHPpxaRlwXwOv51wnhvxNHoesG+ktzcKzZ8tsYFc9RmgD2u1/aGhsUZYdM8tZBgqBwf1rlvF/xOg8SWJii0/7PL1DKOf5155u6AHvTvvEnuKAPUvC/wAaZNN01bPULVrxIxgbsH+tQ+MPjJca7ppsLKA2sBGCorzPGT0z9KVkZD8ylT/tUAdl4O+Ib+FLC5gVMmYYJrI0fxNLpGuJqIXLB94FYXVcdqXceueaAOk8c+Ln8Zakbt12k1zGwU/JpKADHbrVzStQm0m9juIG2vGc1Tpck0Aett8dvtlokd5p4mk24LN/+uuBPiOL/hIP7Q8g+Xu3eXxisGkZdylRQB7Npvx+h0xlMOmCEjrtA5/WqWrfGq21CNimmBZG53YGf515L/EO2DTqAO81v4qTatqFjcNEwFq2Qvr1p0/xSlm8URax5PzR9s1wHToM+lDI2DlSuO5FAHYSfEa7XxI2r2+YpGbcVz1ruo/jxaXMYa80vzZsfewOf1rxX1/u9qX0oA7Hx94+HjCRfLt1giHYVL8NfiVL8P5pZIYjIJOMCuHCnOTzTqAOg8beLn8YaxJqEq+WXPQ1z/45oooAKfb3D20omjba6ncDTKMcY7UAeuaH8dBFp8UGo2n2kxjhj/8ArrE8efFm58WW62sEbW9svGwV597dqTB6nnvQB614O+OsnhXSUsfsTMFXBbjH86vv8frS4B8zS1JYfex/9evFORknOT2p6q3TbjjtQI3L7xNI3iA6raL5D79wVa9Ns/2gY/safbLHz7hFwHPr+deLbctwCTSsp7q2KAudP45+IF54yvWmk3LFnKpnpW74D+Llx4VgNvcxNcWbDBRq86OOvQ0hyqgYwaAPWfFPxsbVtPez062+yIy9uK4Dw54qutB1iO+hJEqnLc9axWOBnFGR+fSgZ7o/7Qlu8KyHT/8ATGGPN4yD+deUeJPFt34g1r7fM7OVbcN1YXPy0inqO+aAOz8Z/EGTxdZ2ls6bfKGM0aF8RJNB8Pz6SImdZhgkVx6qdwA6ik2FGPrQB23gT4iDwjJKJbM3CyHjOK7W5+O1jbQyNY6b5Fwy43cdfzrxINheBxmlKMF6EGgC9rWsTa5ey3dw2WkO41QJ6EdBS8dMce9BXjGKAFopFB79KWgAooooAKKKKACiiigAooooAKbJ91vpTqbJ91vpSlsOPxI8N8Vf8h2b61jf8tK2fFX/ACHZvrWN/wAtK8Gp8R9dR+BDW60lK3WkrJ7mwUUUUgCiiigAooooAKKKKAFbrR3NDdaO5qZbCfQktf8AXxf739a+o9F/5Blv/uivly1/18X+9/WvqPRP+QbB/uivbwWx8/mu6NOij/CivZPlQooooAKc33TTac33TTNI7H1j+wZ/x+61/u/4V4t+0V/yU7Vf+upr2n9gz/j91r/d/wAK8W/aK/5Kdqv/AF1NcVP+Oz06n+7o81/xoo/xorsWx5HVhRRRTAKKSl9u9JagITVmxsZtSnSGKPzWY4AHaoooTNIFx944+lelw+D18NaTHq8cyl8b9veuDEYhU9I7nr4PBOs+aekSHw/bHwDcB9Ttxtk6bhXK+Ltei1PUJXtT5cbHIC9Ks+KvGVx4lWIScBeK5kqMVjh8Nzy9rV1Z1YzFxhH2NHRDCSzEk5FL7UL0pfevWWh89rcKKKKBWS2CiiigYUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFHUT2Eb7pr59+Kn/I0TfWvoJvumvn34qf8AI0TfWvLxvwnt5b8ZxMn3h9KbTpPvD6U2vAifZdAooooEFFFFABRRRQAUUUUB1JYP9Yn+9XvWgf8AIIt/oK8Fg/1if71e9eH+dIhz/dFejhTx8fsaR+aQ454q3Fot5cIWS3kYHoQK2PA2kwatr1tFO+2JnAI9a948Zao3gkJb2Wj+fbKo/eKuf616h4C2Pmmazmt2AkiZD7imtbyxx+YyME/vY4rs/H3i638QzW6pZfY5lPIxiu41vRrT/hVNtcpAonMeWfHNBR4pFbyTEiNGYj0FEdjLcZ2Ru30FevfA7SbXUvt6zxrKVUnJHTgV0Xw6s7Cztdaup7dJRASVUjpyKAPBG0e7hkVWt5FZugIptzYz2bbZomjP+0K+orfxBo+oeEX1g2EfmQdBjrXF/Fi0tdY8F22tW1ssJZcnaPrQBw3wh8Mr4i8RxeYm6GM5Y9qm+L+rW934he0tEVYbf5Qy967D4A28cGmardj76KSPyFeReJrprrWriRurOc0AdL8NfAUfi64L3E6wQpyxY16Z4u+F/h7TvBdxe2MiTyxr99O9eE2GrXdipjt52hz12mvdNNkM3wamZ2Z5DHyzfWgDwXT9NbUNQjtYUJJfaBXv8PgHwn4R023XWdpuZlGQ3UGvLPhXCk3jS3jddwMma9k8V+G5/EPjxJp1K6fanLZ6dKAOK+KXwss9I0dNV0sZtyNx46VV+FHw1tNWs5NU1VglrGM8+leseJtYtfEngG9ihQCOFdv8qwNLspY/hSkNqMyyJglfrQAyH4e+EvGVndx6Mu24iUkNjqa8cs/Adzd+LBpRDFg+04r6D+HdraeC47TT2YNfXH389e9Zel6fGvxckyoA3E/pQBnP4J8HeGZodNvtkl3LhTkcg1wfxZ+GcXhO4juLJS1tLyuBwK9EXQEvvHV9rWonba2r5VX6HpWp8ULiDxH4Jiu4gMHG386BM4P4f/DjSrbw+db111SAjIDVs6p8N/D/AIu8PzX3h9VDxKSQorT8YaXPeeD9K0u0RglwoBK9utdb4G+weHbYaDCFeXy9sh96CT5JMbaTqe1hhon5B9q9g8UafD4u+HsOqQRDz7ePLBev+ea84+Jdqlp4ovUT5dshr1P4NS/2l4P1S0nG6PYf6UFI8FZSrEHrmkqxqA23sw6fOf51XoGFORDI2ACT7UnpXtnwP8B22p28up3kPnJEMqpHWgDx9tHvFj8w20gTruxVVIXkfailm9MV9NWmrXGoambCbw8y2LNsEhUYx+dcB8RNDtvAPiqG6igXyXfJTtigDySaGS3bbIhRvQ09bOdofNETGP8AvY4r3nWvBNj8S9Ntr3SFRJgB5gUdKz/iHNpPg7QV0OCOOS827XcDkGgDxSKB5n2IjM3ooqxNpN3bx75LeRExnJFe7fCvwXFZ+GW1mW0+0yldyrgV0ehXUnie6ex1DQTbxY2h8D/GgD5a6rnt0qzBp9zcf6uF2HsK9C134di3+I6aWi/u3k5X2r0Hxf4k0j4Yxw2NpYrcXOMMVAzQB89XFjNaf62No/8AeFNa1ljj8xo2Cf3scV6J408eWHizTUQWq21xnnium1/SLWH4Q2lykKef5fL468mgDxSG3kuM+WhfHXAqexsJbyZFSNnBODgV6l8B9Pg1RtSW5hWXapK/kKwdG8ZR+EfEF1CbPzw0mFyOnNAGp4v+Fi6J4ZtL+JS0siZZQOa8yuraW1wZY2TPYivqnx548j0rwjaXb2YlDLnYe1fO3jLxYniiVZI7f7MP7q0AUvCdvu1KCSeFntwRnjjFbvxOuNPm1UHT4wkY67R7Vu/CnxpBatb6XJp/nljguQP8af8AHixt7TxEBbxCJSPu44oA8sjjaZgEBYn0q2+i3kce9reQL1zivWfgv4ZsPsN3q9/GsscC7grD6Vu6L8T9G8S602mPpkccLNsRsf8A16APn0KS23Bz6VaXSbt22rbyFvTFemeK/B0Og/ESCJV/0eSQEAelejfEjxJo/wAP5rUR2KyTOBxgelAHzRcWM9qcSxNH/vCoPevoHx5Zab4t+H8euQW6284jywxXgDqFbg5AODQA2pLe3e5kURoznP8ACKiYA4ycV0PgvxOnhu+817YXHbGKAO2n+FKL4AXVQrfaSu7ZjmvMbqxntWPmxNGP9oV9Vy+PUPw0/tA2YC+Xny8DHWvn/wAbeOovFEaolp9m29WA60Acrp8Dz3MZWNpFDDO0V3HxGk0qSztUsIQjqn7zAxzVLwD40g8OyLBJZi581goZh0ruPjvZ2sdnY3EEKwmZMkKPrQB4xBZzTqTHEzL7DNTNpN4jBWt5Ax6cV7P8GVsrPwzqN/cW4m8pdwBH0ruYfEekXfhE6ybBN6c4wKAPlm5tZrPAnjaM/wC0Kdb2M90cRRNJ/uivefiLpll4o0PT9QtYFiecjKqPrXRyeH08BeH4ha6Sb25kUH5QMj9aAPmG40+a2Y+bE8Z75FRKpyFAyT0r6dm8Pp468N3LXemfYLiKPdlgM1886dbxaf4hijunzHHLj9aAM6WznhTfJEyL6kVACD0r6G+LeqeHH8E+Vp5hafywPl65r55BzyKAFpVxkZ6UlFAHrfhBvAw0uEakyfacfNu616B4X8G+B/EzSNaIJPLyfl+nSvmSNSzAY+YnGK+hfBkP/CCfDOfUpvkuJovlz1zQJnBHTNAj8cPHLIsVhG5Bz0rtoI/h3cypbIyln+UN2zXh91dNe6hJKXIMjZr1z4c/CvTtYNveS6nEsnD+WSc/TpQSY3xR+Hdr4Tmt7q3+ezlOePStrRX+HrabGbxkafbg/WrP7QeoSxxw6akLrFGu0Pxg14hZ27XF1FbpuLO2MUFI+ldH8B+CvEOiT3dqimONCd3YV5P4R8DweKPF0lpCMW0cp+btivQ9aZPh/wDDOO1B2XFzFz61n/s/x7rXVLo/6xUJDe/FAzei8F+BptS/sVApvenA5zXkvxJ+H8ng/XvsyAiGRv3fFeoeAbGPw9eXPiDVuSz7ot/XrWv8XI4dcvdEvCgwxB+vWgDnPD/w78P+G/D8F74gdPNmXIVutU/iB8MNLk8LnWdCX91jdkDoK6b4gaDceItc0uwQFLODG89sYJrqbXUNP1Xw7eaPaopit02H8qAPBvhP8Nl8XXUs14cWsPLt2r0u38DeDPE082m2BBuY1IDKOppfAsDWPgbUWto8SOCvy/Wpfh3ZxeA1Se/5u7k/Lu69aAPBvGHhWXwvrk1k244b5RjmsmLS7yVigt5GcHptr6B+Jmn27/ETT2aMESMCRitP4ieLNH8AXVusFhHJK+MjFAHzTcWU9owWaJo2PZhUUdu9ywSNGdvRRX0B8SNPsPFngWPXILZLZ9m75RVX4M+HrL+wbq/FqLu5jXIXHPagDw+bR7u3QvJbyqo7lar7dvHcV7pq/wATxZNPb3+iskYBUZUY/nXiuo3AurqV1TYC2cUAVAQelLSZ5NLQAUUUUAFFFFABTZPut9KdTZPut9KUthx+JHhvir/kOzfWsb/lpWz4q/5Ds31rG/5aV4NT4j66j8CGt1pKVutJWT3NgooopAFFFFABRRRQAUUUUAK3WjuaG60dzUy2E+hJa/6+L/e/rX1Hof8AyDYP90V8uWv+vi/3v619R6H/AMg2D/dFe3gj5/Nd0af+FFH+FFeyz5UKKKKYIKc33TTac33TQaR2PrH9gz/j91r/AHf8K8W/aK/5Kdqv/XU17T+wZ/x+61/u/wCFeLftFf8AJTtV/wCuprip/wAdnp1P93R5r/jRR/jRXYtjyOrCiiimAVNaWM184ECM7Z7CnWVjLqFxHHCMuxwK9B0Szb4fzpNqNsHDjIzXBicSqcbQ3PXwWDlUlzVNibwnpOj22jyjUSqXQGfm6g1xuta3cuHtknZoFOAM8Yqbxd4gTVtRkntx5MbH7q1zjuzdetYYbDuf72pudmMxipr2NHZCHlqWkpa9ZJRVkfOSk5O7EWloopAFFFFMAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiijqJ7CN9018+/FT/kaJvrX0E33TXz78VP+Rom+teXjfhPby34ziZPvD6U2nSfeH0pteBE+y6BRRRQIKKKKACiiigAooooAmg++n+9XvGg/8gmL/dFeD2/30/3hXvGg/wDIJi+lejhdjxsw+BHR6DJcx6hE1opaZTldtey6H8YdWWaOx1TTZLhSQpLLnArxrw/rkmgalFcohfackV6jD8c7dYlc6apm9SB/jXqHhF/45eHNNtbO01CCEW8k2G29xWlfWs2ofB+IW6GUiP8Ahrybx14/vfGtxHJPxGnAQdq2vA/xeuvCtibGVPPtTxtPagZ3PwF0eexstRmmgaMFTjcPYVL4YJHh3xEV9G6fUVys3x6vLfzI7S28qBuu3Fcvp/xKvNPtb6BVyl0Dv9v84oA9K0UFvhXeY6YPP40eLcH4PWIH/PPn06mvNbX4hXNt4bl0jH7mQfe/GlvviFc3nhdNGdf3cYwDQB3H7P8AepINR09nAeVSFHfoK8x8ZWUmn6/dxyLtAc59cVN4D8SP4X8QQ3qybUDZb6V0Hxb1TSdc1L7dp0yvJIMsooA4e0s5byZEgQyMTztFfQWk6ddf8KfkhMLed5XCY968R8K+K28LX32gRedzwDXokf7Q16sPk/YwIgMFD0/nQBy3gOR/DvjK0a6Tysv/AB17x8YNem03Q0XTomma6X76V88+LfGzeJ7yK5WJbd1OV212Hh344SWdgltfwLdeUMKW5oA69t/hX4TyveKVnnTO1utdL8I9TgvfAJVlEkkKFtvevCfHPxSvPGf7sjyrZOAg6VV8CfEm/wDBVwPLPmQtw6HpigD1XwPJqHiL4hS6ldRPDbW7kjd0xip9B8TQTfF6TcyhGlwD+FclrHx0e4sXhsLZbd2GGZf/ANdeZxa9cW+pi+RyJd24NQB9B/GrULsXkWm6bbybJmwXTv8AWoPiBcL4b+HdhZTMBNtGV71y1j8fHFmgu7NZp0HDt1/nXn/jTx9e+ML0tcZWIfdXtQB9L6br0U/w7hv4oxNPBFlVX1rlfhF9sutQ1PWdQDQpyQZPoK8r8C/Fi88J27Wrx+fat0Vq1fFPxsudU01rSytxaLJ97Zx/WgRxPja/GqeJLyQMGUyE16/8NU/4R/4c3+oSHaJY/l3fhXhli0cl4jznau75mNen+OvHVj/wjdppGkzKYwuHC0DPLr2TzLyZgc5YmoaN27BJyx60UAKPusO5r6Q+CmrPF4Hu4rYg3IjyFHU9K+b1GTycV0HhTxrf+EZvMtJWA7r6igDvrz4reJ7W+eJbNlKvtxtrmPGWs654u1CFL6BhMxwFIrpv+F5W7QK0umrJOeS+B1/OuUm+Js15r66lNApKnISgD2nwCbb4XeG4jqMuyS5H3Sa4X40+EJZL7+27Uma3m/ebh0FcF4y8fXvjGdJJiYkT7iA8Vp2vxUvP+Ed/sq4Tzotu3c3agD2Pwjr9zH8KE/s7ElzDHzGOuc1wi/FjxR9oZFs3Dg88c1xfg74lX/hGY+QWaCQ8p2rsn+OkLxll09RMw5bH/wBegCr4d8VXdz8SLe51VfKlZ8fNWh8dvDN42tLfQxvNDLyrKMivMda8TXGq6s2o/dl3ZAXtXfaH8criG1S21C3F3HGMDdzQB55Jo95arHNNC0aMerCvcdWtpNQ+DVsIEMzCPBC15p44+Jn/AAlEPkQ2v2eJehUCrvgb4tXHhqz+wTp9otugVqAOz+Aeg3dra6nPNA8WUJG4fSvHdYUf8JJP6CXOfxr0OX48XEPnJaQCGN/l+WvLL2+a7umnLYZjk0Ae/fEuzkv/AId2DwJ5n7sH5evevn2a2aAKJY2jPbcK9C8P/Ge/0fSksZYftMS8ANXP+K/Fw8R3McggEBQ5IAoAtfDezmTxDaN5T4Zh82OK7b49abcTeIleKBpBx90e1YWhfFw6LbwRpaIZIh97FaGqfHifVmZ57RHLcAsKAOq+FNsL/wAB6lYJxOU6fxdq4PwP4V1AeNI4mtpFVZeWxwKyfD/xIvvDeqm6tiBGzZZO1dnN8fGWEm3skW5I5kA7/nQB0PxMu4m8fadGm15FYBsdqxP2istqtj7AHJ+leayeMru714arMxeXfnFWfG3jy48YyRSTDbsGBQB6uoP/AApFsjnyev414LINrkDpnNdb/wALHu/+ET/sQL+7xtzmuR6lietADMdO+KdH99eOppKRjt59KAPoVbd7/wCC4SAb5Fi5A69a8Cu7WWOT97GYgOu8V2ng/wCK994ZsTaALNAwxtbpVDxh46HihFQWywEd0HWgDC0exnmvLeSKF3TeOVHHWvZ/jfZzXWl6YI4WchB29jXBeGPih/wjunrbrYrMy/xEVv3fx8ur+NRcWSvt+6COn60AaHw3hkt/h/rCSKyuE5B/CtHT8N8KZsDHy/1rgX+Klw9rd24h8pLjjiqVv8QrlPDkulFcRsvB/GgD12S6S38A6KzDaeMt+Jra+K3i3WNLsbK502L7RCyAZXnFeE6h8Q7m88PRaVt2Rx/datnwz8ZL/TLRLO8X7VABgbuaANWP4oeKb23lRbVkjZcO2K81toZNb1xYXk2yyScn3r0LWvjMlxZS29nZLCrjBwK83s9Se21JbvdhxJuC0AejePPhDfeEfD4v57nzUYbtozXln6V6L4y+M194w0X+zpY2SLbt5rzqgAooooA6DwHoP/CQeIba3CEq0gJr1H486xHp9pbaNbfKsa4ZRXlvg3xc/g++FzDH5rA9areLPE9z4q1SW9uGwzHO30oAradpd3qjMLWEyuOSBW7o0Wv6fqECQRTqVYZGOnNVPBXi6Xwjf+eiCYN1DV6Kfj1BHCWXTUWYj72O/wCdAjpvjPGkngu1mudv23Z/F1rzX4M+HB4g8VWxZCYozl2x0rI8ZfEDUPGTqbp2EYPyp2pfBfj6bwWJvIT55B1oA7H9oLxEL7XFsLdsxW527Qfatb9m3UoVvriymxiXsfwrxrVNVm1jUprqdv3jtnmp/DviK58N6lFd2zlWVucUDPcvH7X2seMbbR7a1dLON/4RwRSfF3XIdF1HQ7MHJiKhvbrWHJ8f2ktxIbNftYH+s7/zry/xL4ru/EuoNe3LktnIz2oA+ofHmtG28FpqGnxedPLHwy9a4/wOs/h/wPqep32Y3nUsN/XtXCeEfjZc6Lpi2N5B9rt4xtG6qPjb4tXfii1NnDD9ntDwVXgUAeu/AvXLbUtDvbR8PIuTtrBRtQ8V/EhYZIpIrS3k644AryLwV40vPBmoefbOSGOWX1Feh3nx4zbt5Fosdy4wZB1/nQBt+MtUW++KFlEkgby3A+WsH9of/kPW2OxA/SvOYfFdyniJNUclpQ+481b8Z+N5/GF5HcTrgJ0oA9cvGx8E0B5Hk/1rzvwJ4s1vwuGns4JJrZuoA4qg3xGu5fCY0QpiPbjNXvBfxN/4Rey+zy2onTHpQB674N161+JVrcxajpnlsqnLsvFeAeLtMj0vWLiKFwyCQgYrvNS+OMn2OSLTrQ2pdfm24Ga8svLx7+ZpJS25jkg9aAIuaKKKACiiigAooooAKbJ91vpTqbJ91vpSlsOPxI8N8Vf8h2b61jf8tK2fFX/Idm+tY3/LSvBqfEfXUfgQ1utJSt1pKye5sFFFFIAooooAKKKKACiiigBW60dzQ3WjuamWwn0JLX/Xxf739a+o9D/5BsH+6K+XLX/Xxf739a+o9D/5BsH+6K9vBHz+a7o0/wDCij/CivZZ8qFFFFMApzfdNNpzfdNBoj6x/YN/4/Na/wB3+grxb9or/kp+q/8AXU17R+wd/wAfmtf7p/kK8X/aJ/5Kfqv/AF1NcVP+Oz1an+7o819frRQe/wBaSuxbHjdWBOMcE81LDCZJAvbPan2djPfSbYIzI3tXovg+w0mz02dNTVYrvHG7rXDiMSqatHc9fB4F13eWiJYfCUeg6CmrR3CmRRuC1ynijxrdeItkcxGI+Biqmsa5K0zW0czNBuwFzxWLkt8x4Jrnw+Hcpc9TVnbjMYox9nS0QZw3PNJ9eaSlr12fO76hRRRSJCiiimAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUdRPYRvumvn34qf8jRN9a+gm+6a+ffip/yM831ry8b8J7eW/GcTJ94fSm06T7w+lNrwIn2XQKKKKBBRRRQAUUUUAFFFFAia3++n+8K940H/kExfSvB7f76f71e8aD/AMgiL/dFejhdjx8w+BI0qKTPWr9tol7dIGjt5GX1xXqHhFGipbi0ms22zRtGf9oUjW0qRhzGdv8AexQBHRUkNtJcZ8tC+PSpIdPuLjd5cTPt64FAFeirsmi30bhTbPlhxx1qG+065sTtniaL/eFAEKxtM4VRuJ6Ad6tX2k3WmqjXMTRBum6uw+DvhtPEHiSLzU3wxtuPFWfjJrUWqeIpba3iWOKA4XFAHnX8PPzCjB2lh0Fdz8Nfh/F4umaS6nS2t0PJc16d4u+GHh+x8D3FzYOssqL95aAPnjncOOg5NAU84BJ7Yq3ZabNqF8lrErMzNt4r32PwP4a8FafapqxVrqZQCMdDQB878qvPPr7URqWX7te0fFf4X2em6LHq+lj90RuYAVS+Evw3s9W0+XVdTOy3j5+bvQM8l8souSDj1IpBwpJOfevpC28EeGfHVjcw6YnlyRAgNjg14/Y+AZrrxb/ZGDt37c+lAjj1DNtwpPqQKRvk3DtnrX0jJ4V8IeE5INJuVE12+FOK4b4wfDGPwvIl3ZgfZphnGOlAHk+75gPbtTtpCgkHHqa9q8AfDfSrHQDrmsFWg27tp61vXnw50DxtoE17pKeV5aFsY60AfO6gt93k0vlt5hHzAYrtfAvgRvEnif7C4Kxxvhm9q9eh8L+DZdUTRFUPdY2lgOM0DPmlcMCAMYq/baTd3lsZ4YWeCPqRXW/FjwMPBOriNP8AVSH5T6it/wCCd9BeXkmj3CKyTjAJ+lAHk+O3ShuG46V0fj3Qf+Ee8Q3FtjCqx6VzYzz9aAGsu7FLtFLQOe9AhNo9KNo9Ku2uk3V6uYYWceoFRXFnLaybJUaNv9oUAV9oo2j0q1/Z83kiQowTGc4qKG3eZ1VFJY9gKAIto9KNoqWSB4ZfLdSr+hq1b6LeXShooHZeucUAUNo9KNoqWa3kt2CyIUbOMEURW8k5xGpds42gUARbR6UtaE2g31vHvkt5FXGc4qrHaSzA7FL464HSgCDaD1FG0VK0LLJsP3qfJZzQsivGyM/QEUAVdvQ9BS7Bx/OvUfhd8NU8TW1497GyFFyu4Vw2teHrnS764TyXESOQGI7UAZB5GKX7PIV8wI2xe9SwW7XDYjG9v7oHNei2s1hb+B5Lea1IvAv3iKAPMMbeByKKkkh+fPTNWrbRby8UmG3eQD0FAFGpaddWU1nJsmRo2/2hU0en3EkZdYmZfUCgCvRV6TRL6OEStbOEP8WKo7WGcjvigAprY6mnUnG7rzQMNvfOM9qQDPXmu++F3gP/AISjVFiu42WNujEcGsvxh4Pn0PWri3hhdolPDY4oEcuPpinLC8ikqjMF64oW3aSTaASw7V6L4LksdN0O7Go2rGRl+UlaAPNwu3ilqaWMzXDBF53HC1MujXjZb7O4AHpQBToq02lXghMv2d/LH8WKgSF5H2qu7PTFADKKvtod8sZkNtIE65xVAghtpGKACirEGn3NzHvihaRfVRUMkbRMVYbSKBjaQ9uMjvS0dePWgAx1wuBSAE8hcr6gVveEfD0vibVraziG5Xfa1e133hzwd4Dt1tb8LNdkfOB2NAj532lV2n659Kb8zc7c49q9C8fP4fmureTSwqRMfnFdXpNx4Gj02FLmMGbHzHigDxNULH7vH0pdrNwF5r6d0Twb4N8QaLd3ltCoWNCQx6V5b4Xm8MW+rXK6ioEW8haAPNPLYkcHC+1Iy7SAVwOvNfReh2vgXXtQW0t4Q0kh44rk/jN4Y0Pw/q9tBaqqgth/pQB5AsbEIAuQ3cUbT9CK9w0/UPA2n2MKyRCRioB9jUnjb4e6HqfhltY0cqqKu4qKAPCW4+b7vrT9jg5KkDHXFek/CP4cx+KbqWW/+W1gOWZuhFelab4S8JeKJrjTbNP30Y2hscUAfNa9BgYpdpPTP4V2Ws+BJbHxc2lAcGTaMDtXq3/CFeFvBNrbRapiW6mwMUAfO33RnFN3Hgj5l9a9p+KvwrtNL09dW03i3cZC+lVvhZ8OLC/02XV9VIW2jXd7UAeQgEdQSPegZxX0fH4E8MePbC5GlKI5IlJGB3rwLUtFnsNSltCjb0YgLjmgDOoq5DpF5cErHbuzDqAKguLSa1kKTRtGwOMNQBFRVi2sZ7xgsMTSE/3RUt5ot7p67ri3eJf7zDigClRRRQMKbJ91vpTqbJ91vpSlsOPxI8N8Vf8AIdm+tY3/AC0rZ8Vf8h2b61jf8tK8Gp8R9dR+BDW60lK3WkrJ7mwUUUUgCiiigAooooAKKKKAFbrR3NDdaO5qZbCfQktf9fF/vf1r6j0P/kGwf7or5ctf9fF/vf1r6j0P/kGwf7or28EfP5rujT/woo/wor2WfKhRRRTEgp/8JplP/hNBaPrD9g//AI/da/3T/IV4t+0V/wAlP1X/AK6mvav2D/8Aj81of7J/pXi/7ReP+Fnarz/y1NcNN/vmevV/3dI8z/h/Gp7KylvrhYIh+8Zqjhha4YKik5PavS7fwfb6ToEeqpOv2lRu296K+K9iuWOrIwmCdZ889IjdJ0uX4fzJd3sIdHG6uY8XeIBrOoPPCNit0UUviTxlda/GsMzcINormiueSee1c+GoXl7ap9x24zFqnH2NFiMaKSjmvX9D51+9uxaKKKVmISk3U6inYBN1G6loosSJuo3UtFFgE3UbqWiiwCbqN1LRRYBN1G6loosAm6jdS0UWATdRupaKLAJuo3UtFFgE3UbqWiiwCbqN1LRSsA3dS0tFBVwooo/GiwXCik/DNHTrxTswFooopdRPYRvumvnz4qf8jNP9a+g2+6a+fPip/wAjNP8AWvLxvwnt5b8ZxJ7UlKe1JXgRPsugUUUUCCiiigAooooAKKKKA6ksH+sT/er3rw//AMgeH/dFeCQ/fT61734fP/Eph/3RXoYU8fH7HaeANLg1TxJbQ3P+qZhn0r3fxlrVx4LkhttO0kTW4H31WvnXRI7x9QiNkrNMrZAWvZvDvxT1mG4isdT0qSfkLudf/r16p88ef+OvFUPia8i/0D7PIDyMYru9b0e1X4T29ysaLK0eS2OetT/HXQNPjtrTULeNLeaTBKjg1evrOXVPg7bR2qGV1iwQv1oGc/8AAvS7bUGv1kiSYKCcsOnAroPh7FZ2FrrV1JbrKYiSARx1FRfAfw7eWdrfTSxNFuU9fpTvC7A6H4iHPAbn8RQB0kHiHT9S8Jvq8lgivb8gY61yXxSt7PXPBNtq8VssTMmflH1p+kOG+FN7gHof50niaMn4N2JI48vH6mgCr8BbYW2k6rdIMuqEj9K8k8T3LXGrXDtyzOST+Neq/s/6hG0eo2DNiSZCqg/hXmfjezfT/EFzBIuwoxB96AMyw1e708lIZngVjyq9K9601pZPg5cySPvby8nNeBafYTX9wkcETSt3C19FabpF5/wp2a3MD+aY8bMdaAPI/hHbrdeNrYSnI8zpXsXijw++u+NlmvV2WFuclm6cCvHPAEkvhvxfaS30TW5MmPmr3r4xaxcWuixrpkJkN0vLxigaDxNrFp4m8AagsIBjhTaNvTtXOabazx/CrybNC7yLgbfrUUyv4V+Fc3njZNPHkqetdH8G9egvvAbxhFmeBD8hoKHfD+3tfBb2mmbV+23A+b1rP0qyij+LUwCjLOSDj2rG8DrqetePpdUvI5I4IXJy3QUmkeLoF+L3LjYZMA/hQSzRj8PJeeN77VtUO20hfKF/wrV+KWoQ694EhuQoZAPl/Osz42X19Jd2+n2FuwjmOC8Y61W+IVyPD/w/sbKY7Z9g/d0CLXi/RpbzwfpGn2ynZOBuVfxrrvBE1ho0H9h22wuE2yD8Kp6VriXXw3j1KCPz7iKHhR1BrkfhNDeG81LWtTVoE5I39+lAF34daXFb+ItfkjjxJGzFSPoKoeB9PTw/dXeu6qwWbeTGj9TzU3wn8T2d14y1GBnys0hH8qj+Ji6jrHiy30q0tHS2Rx90cGgpEH7QgTU9BsL8oAzLkfrXj/w61CW08V2UiEq28DivTf2gr0W9lYabuy8SgMvoea4L4RaQdS8WWfG5UbJNAzqf2hLBLfWopwoDSDNeQ7twz616p8fNaTU/Efko25YflxXlf+RQAV0Xgjw//wAJFr1rakZVm5x1rna7f4Q6gmneNLJ5GAQuMk9qBM9s162k8D2lvaaXpLXDbfnO0f41m+KvCkXiTwTJqU1n9ivI13YYc0/4veKNe0nUxLZQtLaSD5XUZ4rzzUPG3iy+0eUNAy2hGCxFBJqfC2+0vxFZzaHeiMT8qrEc10uk/Duw+HcF7qmqhCBlo1bvXGfB/wAHS3up/wBrXjeRbQtuZuma9R8WT2PxT0K5stPm/wBJhXAUHqRQB5J4P00fELx0H8jZbmT7gHavU9cubnw7eLZ6bovmwrwzKo/xrgPgrjwx408i7zHMrFQGrZ8e+MvE2i6xLHHbs1uWyr4oAf8AFbwbb6h4Xh1dLX7NcBMsuO9R/D3w3p/hfwY2vajGJG2bkVhXI+IvF3ia/wBJUXkbJZn16Gu9toJPGHwhS2tD5lxHHyq9aAOaf41W135sMunJ5RBUcdvzqb4K2trrmvaiWhDxyZIVhwOleX/8IvqUlwUWzlLDjp0r1D9n2NrfXLmFwUkII2mgDjNYtI08dGM48vz8Bfxru/jTa2mg3mmy20CDaAxGPY1gal4S1Cb4lFBbSbFnzuxxiuk/aPjMElkrjaVUDH4GgDr/AIPePl1qyu2+zLEYk6KOvSvL/G3xSF497afYlHJUNitz9nkiS31BF+ZmQ4X8q8z8WaJfWurXRltpFXeTuxxQBV8M6/8A2HqX2sx+cDzsIr3C4vbXxN8NZtSa1jgl8vPA5r57tbWWdikcbSH/AGRXv2gWM/8Awp+SJo28zy/ukc9aAPG/B+kLr/iSG2Odrvg+3Ne1+MvFem/DWS206zs0lkGBIxHtXl/w63aN44s5LiMxoZOrjFdX8dPDV7L4k+3QQtLDcfdK0AbPjTRdN8ZeB/7btYliuRHuIAq18MINMtvh1Lf39vHK0aZyw96TT4X8L/COcXw8t3jwqt1ql4Zk3fB2+GMLsI3D6igDV8C+NNO8eXVxo01nEq8qrAV4d8QNLj0bxJdW8PCK5xXb/AQ7vGSgHI3H+Vcx8WFz4yvcjGHNAHHK27606N/LmVg2dvJFM2470Yx75NBZ9F/BLx0NQvIrNbVEKjG8Dmsn4jfFQWuq3libFWbJXfisX9n+RR4kAOAc4A/CsT4p6Ler4uuybeQoXPzAcUEs5ez1g2esJe7dwLbinavoLwjrNp4x8J6jLJZxxGOPAIH0r5uhtpJ8KiFm/ujrXvnwbs5rfwfrCyRshKHaCOe1Ajy7wZaJN44ghZAyGXofrX0F/a2mp4tGi/YYijfKcD2rwvwTYz2/j6B5EZF87qwr1S4b/i7inIILdvpQBr2txp2p6rqmiCzVUiyA2PYVy/w1+G1s0l9qU1uJxExKoRWp4V/efEDWwB91if0Fa/g3V/8AiQapbWjr9ujztj79qAMzT9WvbjVja3WiYsGbYDtGAPzrzH41eCrfw3rqSW+EhlO/avatO4+IHiqG9aD7I/mK+AAK474ha5rWqXES6qjxyDgBqAPTfg7qvh+08OTrqTQpKyfKH61474wmguNdufI2mHecba7LwL8J73xho73dvcmNVXcVFcBr2lPpOpzWruGZWwTQUUqTn0zS0LjcKAZ65+zjZpceLUye/wCVejeNtD8LNr8o1W+U3Ltja3avNP2f9Ui0/wAWRpIQhc8ZqT4v6Jqtx43kmjgkkRpPlZelBJX+KPwxi8NxpfWMnnWMi7lbsK81s42urmOBBveQ7RX0L8RGFl8LbS1ueLkRdG6968z+Dnh1fEniqFtuY4m3N6UAd/r9yfAPwvisUxFdzR/N65rwGSQySM/3yTmvVPj14lTUNaFrGfkh+UYry2zha6uFjRcsTgCgD2P4A6GjXFzq10gRLcbgx/CuG+JfiBtb8SXcpmyqsQK9dvFj+HXwp2N+7urqHle55r5+ghm1e6fZzJI3A9aAO9+G/gnT/FEbNdXyQ+gY16l45sYvBPw9e0sCLtJUwzL0FeD/APCP65pUvliGeNuvy969v8M2V4PhjeNrSniP5PN69qAI/hmr2/w0u5IlJnkQ/d69aueAbWHwNZxz3e37fcnIB69am+AmrWl5o99ZjDPHnC/lWCseqeJvicpa3dLOGTjI4AoA3/EumxXHxU0yXj944Y/kaj8UeFW8RePHmvBts7Q7st04rO8SeKbaP4pWXz5EUgBOfat/42atPHpca6ZCzNdDa7x96ALvi7VLTxF8Prl4Ix5cY2gDpxXOx2Mv/CqRb2akTTDAA+tR3zHwv8I/Iuf3dxKmSp610Hwt1yC+8Ar8olkjj+73zQA/4eQ2HgtbbTcqb24X58dQf8iuJutHgk+NGyRN0Zf5l7HirXw5W/1zxvd6jdwvFBExwz9Kz9K1L+1fjMXRt6CTFAGx408dab4O8RLZWlhG2Ww4xWD8atG0+40mx1aCFYXuBkhR9a5j4vZ/4T+XPH7yut+Kp3eAtJX7o2cH86ANPwRo9vo/gBNTtbRby528qBkiuR8T/EKW80ua1vdKaFiMByvT9azfB/izxB4T0+MxWs1xZt0GMivULW6tPH3g++kv9OFm6ISGYYNAHzKxyxPvSVLqEYt7yVFPygkCoV+6MnJoLFpsn3W+lOpsn3W+lKWw4/Ejw3xV/wAh2b61jf8ALStnxV/yHZvrWN/y0rwanxH11H4ENbrSUrdaSsnubBRRRSAKKKKACiiigAooooAVutHc0N1o7mplsJ9CS1/18X+9/WvqPQ/+QbB/uivly1/18X+9/WvqPQ/+QbB/uivbwR8/mu6NP/Cij/CivZZ8qFFFFMEFO7U2n/w0bjPrH9g//j91r/d/wrx74/WM9/8AFTVUgjMh808D8K9e/YRnSHUNY811QMMDP0Fc18X7eTwz8Sr7VDH58MkpPyivHqSlCdoo+lw6pTgud7HAeDrXTNPtZ11WIQzAfLu61yGua5I9xNBBOxt8/Kue1XfGGtTa9fmaK2kjTPYVzbWlw3Pkt+Va0cLOfvzRljMwpQSp0noQMdzfzpDU4sbj/nk2fpSmxuO8TflXqeycVax8068ZSu2QUlWPsNx/zxb8qPsNx/zxb8qr2cw9tAr496Me9WPsNx/zxb8qPsNx/wA8W/Kj2cx+2gV8e9GPerH2G4/54t+VH2G4/wCeLflR7OYe2gV8e9GPerH2G4/54t+VH2G4/wCeLflR7OYe2gV8e9GPerH2G4/54t+VH2G4/wCeLflR7OYe2gV8e9GPerH2G4/54t+VH2G4/wCeLflR7OYe2gV8e9GPerH2G4/54t+VH2G4/wCeLflR7OYe2gV8e9GPerH2G4/54t+VH2G4/wCeLflR7OYe2gV8e9GPerH2G4/54t+VH2G4/wCeLflR7OYe2gV8e9GPerH2G4/54t+VH2G4/wCeLflR7OYe2gV8e9GPerH2G4/54t+VH2G4/wCeLflR7OYe2gV8e9GPerH2G4/54t+VH2G4/wCeLflR7OYe2gV8e9Jz61Z+w3H/ADxb8qPsNx/zxb8qXs5h7aBVpwqf7Bcf88W/Kj7Dcf8APFvypezmL20CHrSAjGSKm/s+4/55N+VJ9huNo/duce1NU5B9YppDWjbYGKMB60zr7ivUfBXhWHxZoMsDJ5dwo+ViK4bxJ4ZuvDd1JDcxlcH5TjrW0qUox5jGni4TfKjIooorn6nS9hG+6a+fPip/yM0/1r6Db7pr58+Kn/IzT/WvLxvwnt5b8ZxJ7UlKe1JXgRPsugUUUUCCiiigAooooAKKKKBkkP8ArE+te+aAo/smH/dFeBRt5cit2B716xo/jjT7ewSOSdFZRXdh5KOh5GMjKex6V4d1x/D+qJcx4JU55r0qL47oqru02NpV/jx/9evnv/hP9K7zIaP+E+0r/nqten7SJ431ap2PTvGnj698aXSPcNsjU8L2rb8E/GC78I6f9iaP7RbkY2mvFv8AhPtK/wCeq0f8J9pX/PVaPaQ7h9Wqdj39/j/dxLKtrbrAjDGBXLWPxKu7C0vbdRkXQO415T/wn2lf89V/Oj/hP9K/56L+dHtI9w+rVOx6na/Ea7t9Bl0rH7pxy1OvviPdX3hqLSHX90nGa8q/4T7Sv+eq/nR/wn2lf89Vo9pDuH1ap2PSfBHipvDGuQXg+6rfMBW38Vtb0rxFqQvbB1Lvy4X1rxv/AIT7S8/65RSt4+0v/nutHtIdw+rVOx6H4Q8WHwvfefHCrjP8Vegt+0VqEcHkpbqY+m0dK+ev+E+0r/nqtH/Cf6V/z1Wj2kO4fVqnY9Q8WeOJPEl0lwI1hdDkba6/Q/jtPp+mx297bC7MS7VLc18/t4+0vtKtA8faWvWZfzo9pDuP6vU7Hq3jb4oX/jHEbIYLVf8Aln2qHwV8QL3wWwWA5iP3l9a8v/4WBpXUTKBU0HjKyniZ0bzFXqw6Cj2kR/V6vY921P483d1YvBaWotmkXDMP59a81h1S4t9U+3ecWmB3AiuP/wCE+0tefOU0Dx9pZ5Ey0e0h3F9Wqdj6Dsvj5LHbot1aC4lVflc9j+dcL4x8eX/jC8E1yx2r91K81/4TzS/+eq0h8eaX/wA9Vo9rDuL6tU7Hsfgv4t3nhK1a0Mfn2zcFT0rQ8QfGy61jT2tYIfssbdQvFeG/8J9pfeVaX/hPdLx/rV/Oj2kO4fVqnY7vQfEl14e1RL63b94Dur02P9oa5UeZJaK9z2k7j9a+df8AhPNM3f65cU7/AITzS/8AnstHtYdx/V6nY7zxP4su/FGqSXt3l2c5x1xXa/C7xRpPhPTrm7mkQXZU7FPWvDv+E80vqJlBq1D4otpF3Bdyt0bBxR7SHcf1er2Oo1vVpNa1Ke6ck72JqltPJrmpPHGnRyENMqkdVamr4804A5mQd+9HtIdw+r1ex0+01Ja3UljcCWM7WXkGuV/4TzTP+e6Un/Cd6Zn/AF60e0j3F9Xq9j3nSfjhcW9rHb38C3aIuAW5xVTxJ8YJNYtGs4LUW8J64FeJ/wDCe6X/AM91pP8AhPdL/wCewo9pEX1ap2PW2+KV7HoLabAvlowwStZXhPxpe+FNQN1bynLNuIzXnP8Awnml/wDPdfxo/wCE80v/AJ7r+tHtIdw+rVex6Zq3jie/1xdTjHlTZ3cetdhZ/HqdYEW8sxcMBjJGa8D/AOE90v8A57Cl/wCE80v/AJ7L+NHtID+rVex674x+Js/iqz8jylhh7Jiq3gz4i3ng9wIGJhxzH2NeV/8ACfaX089PzpY/HmlyOqi4T3o9pDuL6tU7Hv118dFktmWHT445WGN2Of51wui+NbrRtaGpQsyyE7iorz6TxpYwFVkbZxwW703/AITjTuvmj86PaQ7j+rVex9A3X7QE0+Hjs1W5Uf6w9SfzrjfHnxGvPHUkL3S/MoxXl/8Awnml55lUfU01vHumf89Vo9pHuH1ap2PSfCfjC78HXZubVjz1UV0mtfGObXLOSGW2jRnGC2Oa8R/4T7TO8ymj/hP9J/56r780e0h3F9Wqdj0vwv4wPha4kkEIn3nOGFdrH+0Bei2NuLdFhx93FfP/APwn2lLk+ehU0f8ACwNJ+6Zl56Cj2kO4fVqnY9X8QfEaXXp4pPIWF4+QyV0+m/HWe3s0ivrdbvy+F3V4GfHmmbeJUFC+PNL7zLR7SHcPq1TseteNvilfeL1WFh5dsvAjXpiq2n/EW603wzJoqDMLrtNeWnx9pfaVaB4+0r/nqtHtIdw+rVOx6J4Q8Xz+EdS+1wjLD0qn4g1yTxFqc15Jw0hzXCr4+0wfelU07/hPtLLAiZR+NHtIdw+rVOx02KTb36mudXxtYyEkSK3rjPFM/wCE800ceetHtIdyvq9Xsd34a8RXPhu8S6gYh1bNdtqXxsudThaOa0Ryw5cjmvDT4+0xes6Yo/4T7S/+ey0e0h3F9Xqdj0Xw94q/sXVjdiFW5ztYcV3tv+0FfWKmOG1jRH4bA4r59/4T7S/+ey0f8J9pf/PZaPaRD6tU7Htd58Yp5pUnS2jjlU5yBVFvipeyeJF1Vh869vwryP8A4T7Sv+eq0f8ACfaV/wA9Vo9pHuH1ap2PXtO+Kl7Y6xd6gg+e461T0n4hahoerSX1vIdztkr2rywePtL/AOey0v8Awn2l7v8AXLij2kO4vq1TsfQUfx7A+aTTVef+/jv+dcJ4w8VTeL74XUwCHOdo7V5z/wAJ9pQ/5arSf8J9pn/PZaPaQ7h9Wqdj2Hwf8WL3wbp72luu5WXaa47V9SfVr6W6cDe5zXIf8J9pX/PVaafH2l9plo9pDuP6vU7HTUfpXL/8J5pv/Pdfzpf+E+03/nsv50e0h3H9XqdjtNM1KXSryK5hk2yxjII9a9Osfj1cLbIl3aLPIvAdutfPw8faX3lWl/4T/Sv+eq0e0h3J+rVOx6l40+Id74wkBlbZGDgR+1N8F+PrnwUs32ZQWkGN1eX/APCfaX2lWk/4T7S/+ey0e0iH1ap2Ox1rU5NY1CW4lbczHdmmaXqH9n3EVztyytnbXIN4+0zPEyD8aVfiBpfeZMfWj2kO4fVqnY9W8afEi78aW8ME3yJGu0Cua0y8bS7mGdTl0Oa47/hPtL/57LR/wn2mf89lo9pEf1ap2PoSz+PJWELc6fHM4GA2P/r1i+LvjJfeJLI2iL9ng6bRXiv/AAn2l/8APZaX/hPtL/56rR7SIfVqnY9H8G+Nr3wdem4tn4Y/MvrXfTftBTfZ2FvZiKdl2mQdf5188Dx7pmcmZcU4+PtKxxMn50e0h3D6tU7Hb3GvXV/qZu5JP32c7q9E0f453FjZpb3lsLryx8rNXgK+PNMDZ85fzp//AAn2lk8zL+dHtIdxfVqnY9W8a/Eq/wDGDbJDshHAQUeA/iRe+CWIj/eQngqe4ryg+PtLPSVRQvj7SwTmZaPaQ7h9Wqdj33WPj1c3FnJDZWotSwwWXv8ArXCeHfF93oOsrqakvKGzXnp8faVj/WrSL4+0vOfOXFHtIdw+rVOx6F4m8UXHiLWH1GXhmOcVp698QrnXNEtdPlHyQjANeVt4/wBLP/LZQPrR/wAJ7pe3/XLR7SHcf1ap2PbfDXxgOi6alnPZrOiDjjrU3iL40z6pp8lnaQ/ZIpBg7eK8N/4T7Su8q0f8J9pX/PVaPaR7h9WqHRyOZGZycsaXcODntXN/8J9pX/PVaP8AhPtK/wCeq0e0iP6vVOk3D1psjDa3Paud/wCE+0r/AJ6rTX8faU0bhZlziplUjYccPUunY8x8WMF12bPrWNuG/PatDxBfR3+qSyIykE1m141TWR9NR+CzFPWkoorKW5sFFFFIAooooAKKKKACiiigBW60dzQ3WjuamWwn0JLX/Xxf739a+o9D/wCQbB/uivly1/18X+9/WvqPQ/8AkGwf7or28EfP5rujT/woo/wor2WfKhRRRTBBTuG47U2igD0T4Q/ERfh/LcOXK+Z6H6V3mofGzRtWy1zD5rf7XNfP9J+NbRlGP2TllSle6kz3L/hZvhngfYk/Kg/Ezwz/AM+Uf5V4d+NJW6xFvso5Hg768zPcj8TvDJ/5co/y/wDr0n/CzPDP/PlH+X/168Oop/WvIPqX95nuP/CzPDP/AD5R/lR/wszwz/z5R/lXh2fejPvT+svsP6l5s9x/4WZ4Z/58o/yo/wCFmeGf+fKP8q8Oz70Z96PrL7B9S82e4/8ACzPDP/PlH+VH/CzPDP8Az5R/lXh2fejPvR9ZfYPqXmz3H/hZnhn/AJ8o/wAqP+FmeGf+fKP8q8Oz70Z96PrL7B9S82e4/wDCzPDP/PlH+VH/AAszwz/z5R/lXh2fejPvR9ZfYPqXmz3H/hZnhn/nyj/Kj/hZnhn/AJ8o/wAq8Oz70Z96PrL7B9S82e4/8LM8M/8APlH+VH/CzPDP/PlH+VeHZ96M+9H1l9g+pebPcf8AhZnhn/nyj/Kj/hZnhn/nyj/KvDs+9Gfej6y+wfUvNnuP/CzPDP8Az5R/lR/wszwz/wA+Uf5V4dn3oz70fWX2D6l5s9x/4WZ4Z/58o/yo/wCFmeGf+fKP8q8Oz70Z96PrL7B9S82e4/8ACzPDP/PlH+VH/CzPDP8Az5R/lXh2fejPvR9ZfYPqXmz3H/hZnhn/AJ8o/wAqP+FmeGf+fKP8q8Oz70Z96PrL7B9S82e4/wDCzPDP/PlH+VH/AAszwz/z5R/lXh2fej8aPrL7B9S82e4/8LN8Nf8APlH+VL/wszw1/wA+Mf5V4b070Z9zR9Y8g+pebPcT8TPDX/PlH+VH/Cy/DO5R9hj6eleHDrknihR0Hen9YXYn6l0ufT3gbxLpmuJKbK3WAL1NcT8ZvEOlzJ9nQJJdAY3CuG0zxt/Yegtb2o2yuuCwrlrq8kvpvNmcyOeea6K2ITp2sYYfASp1LyZFRSUteP1PcewjfdNfPnxU/wCRmn+tfQTfdNfPnxU/5GWf615eN+E9vLPjZxR7UlKe1JXgRPsugUUUUCCiiigAooooAKKKKYC5pePmOSd1GKTb8wNF2ibJ7l3TdFvdWkKWsDzt/dUVp3HgLXrWBJpNNuFjY4+7zX0X+xLp9jdatqM15brcLCCwVhnsK9/8J+OtI8YaHrktzpESnTg2wbR2I9/enzSKskfnZrHhLV9BjSS9spoEbkMw4xS6P4R1fXjiwsZrkdig4r7E+IWt6f8AFL4M6pqRsBbXFmpC7QOxH+NcN8Jfire6P4ct7HSPDDXVxEuGm2defrS5pBZHztrHg3WdCUm9sZoMc5deKxedw54r9FPDel3Xxb8LalbeJfDv9nyxxHZJtGf518xfBv4Q23i/4uvpE677KCc7+Owp80g0PJtI8C67rib7PTbiVMZ3BeKzdS0W+0eYRXlu8EndXGK+yvil8eNJ+DuuP4e0LS4XW2bYzY5PH1rwH4xfFuw+JUNvLDp32KdfvsoHJ596V5D07Hm2l6Df6wxFnbvOV+9tGcVDfabdabOYbmNo5FOCrcEV9KfsW+JvDGhahqbeJJIo4m+4Z/wriv2otZ0XXviPeyaGYnsWf5TDT5pBp2PFhlsAdc4NCnczAcYPGa+ofgz+yvpvijwxDrniG/isLeUbo/OOM/pXJ/Hr9nWb4Z3EV1p7G90+T7rxDI70XkLQ8LwemDmk5711viD4danoOg22qXkLxWtwMoTx/npXJUc0gsg59cV9ffs4+B9M134Sa7eXVujyrCSGccg8V8ggA8HpX3T+yHpk+tfCPWLKBPMnki2ge/FLmkGh8Y32k3OoavPbWkLSuJCoWMZ71UuNFvbG6FrPA6zn+HHNfa9x8PdH/Zt8O3Gva1Ct3q0gLrEw6E18zad8UW1P4iRa3dab9ojMu4W4XPFPmkGhzsXw18STW6zppVw0RGdwWueu7OaxlaK4jaKVTgowwQa+29H+LnivV5LdbPwe62D442Dp+dYH7XXw6sI/Cdn4jhs/sFzcJukjwBg80XkGh8j6bpN5qwb7LA02z720ZrW0/wABa9qULS2+mzyIO4Wvpf8AYR0XSr4awdRgiuI41yTIM+laHjj9qPS/APiW80fSNEhe2icqWVeCPzpc0gsj5An0+5trv7NJCyz5x5ZHNbX/AAr7xB9n88aZcGMjcG28V9K/BrRdL+NXxEvfEstksNjbt5hixweBWte/tQ6db+N00FNHhGmCTyg20dOnrReQWR8bMr2lxslQq6nDK3FfWn7NvxL0TxNcWPhm60NTJjaZmUHPH1rmf2tvhrp+i3Vh4g0uFYra9HmbcY9f8K5v9kmMf8LY05gDy/Q9Oho5pBoYv7S+k22i/EO+gs4lhh34VU4rzeHw9f3FibqK3kaBR/rO1eqftVf8lO1DBwFl4/KvWf2UItI8b+C9U8P3kUb3awlULDkng0+aQaHyII90gQMST096t6hod5pcaPdwyQhhxuFew/Dn4Ttq3xk/seeEG3huCCpHAA5rvv2ktJHivxhb+GfDVksslv8AIfLHXileQWR8p7VJID5+lJtXkBjkepr17/hmXx2wyNDm64yAP8ahu/2c/Gthbvcz6NNFFGMsxx/jTvIXKux5zaeHb+9h82C3lkj/ALyjNX7j4f69Z2X2uWwnWDqX28Yr7J/Z9s9H0X4Nalqeq2KTy28ZJDAZ3Airnwd+MGlfFrWbzw7c6TGtq2Y0+Uen1o5pDsj4Y0/QbzVnZLSCSZ0+8F7VVuLOS1kMcuY3HBVq+n/BMOmfDr9oG50KeFGspJipDDjmuc/aP+Ez+HfiNGltFi2v5AY9o45//VRzSCyPDo9BvpLNbwQSm16mTHFXfC+j3l9qMMkNvJcRRuC+wZ719TfFmw034Y/BPTtIMERv7qDlsfN3/wAK8e+C/wAXh4DultX05LtbiQKWYZxmi8g0Nn9oDVNCutD0mHT7I2VyqASnGM9a8IhhNw+2Pc57KK+v/wBsrT7OTwxoV7b2i2jzRhiEHXOa8v8A2UfCOmeK/H0Eeo4eNG4jbGDRzSDQ83tfhr4luLX7RDpVw0OM7gtYV7Y3Onz+TcxtFLnBVhzX3N8VPitrnw716Wxs/DHmaVF8isicMPXrXyb8TPHUXizxI18tgbFlOWTGKOaQWRz1r4J1q8gWaHT5pI2+6QtLeeCNbsbUzT6fNFFjJbb2r3L4a/tDahJBZaPZaJHcyAhd23P9a99+N/jzRPDPwpFrqNlBDrE8ONgAyM0uaQaHw78P/Bd94o1azRbR5oGkCsyjjHvXrH7RXwDl8Etp76PZzXAkUFiq9ODVP4GfGuTwhq1vp6WKSiaUZZh0ya+jv2m/js/gSTTY109bgzRgjcOnB96d5Bofn7qFve6bdNFdwtC4ONsg5rT0nwfrutxGS006aZQM5VeK9C0O6Pxw+KFlHPb/AGdbiUZVRX0l8WPi1ofwBuLTQNJ0yKSVQElYqMnj60c0g0Ph/UtLvdHlMV5bNDIOCrDHNN06wu9Xk8u0t2mfuFGTX2X8S/D2kfGL4PnxhYWSWt7HF5jhR17V5p+xpo9pe/EpI7uJZot/zKw9qOaQaHimn+BPEGqSNHa6ZPKyn5sLxUcPhDWZNZTTTp8y3bPtEZHOa+zfjD+0Fonwl8TS6do2kRP5bfMQo/xrzLwl8cNG8ZfFvSdVvdPWxhEgLswAXv70XkGh2Hwv/ZVuLXw/dHWZI1upo8hH6pnFeD/Fr4K3/wANZTLNPHPA7HayntX3D4p8F65q8esavpF6LmC4jzEkbdBgV8HfE+38WabcfZ/ECXCRxsQPM6UXkGh57u3dOaNv+c0u75vUdsUYPrRzSFoJt/zmjb/nNLg+tGD60XkLQTb/AJzRt/zmlwfWjB9aLyDQTZ7n86Nv1pcH1paOaQ9BuPrR+dOoo5mPQbt+tG33/WnHNJg+tF5C0E2/5zRt+tLg+tKM0XkLQbt+tG3606ijmZWg3b9aNv1p1FHMw0G4+tH4U6ijmkKy7Cbc+1Jt/wA5pcH1owfWi8haCbf85pdtGD60YPrReQaBt4xRtPrRg+tGD60XkGgmCD3o2/WnUUc0h6DcfWjH1p1FHMx2Qh57Um3606ijmYaDNvuadjjFLSYPrRzSFoJt9/1o2/5zS4PrRg+tF5C0E2/5zRt/zmlwfWjB9aLyDQTb/nNJg54JFOwfWjvS5pIat1E5Gec000M2aTrT8xJWdwoooqdygooooAKKKKACiiigAooooAVutHc0HrR3NTLYT6Elr/r4v97+tfUeh/8AINg/3RXy5a/6+L/e/rX1Hof/ACDYP90V7eCPn813NP8Awoo/wor2WfKhRRRTBBRRRQAUUUUgCiiigAooopWAKKTdRuqg1FopN1G6gNRaKTdRuoDUWik3UbqA1FopN1G6gNRaKTdRuoDUWik3UbqA1FopN1G6gNRaKTdRuoDUWik3UbqA1FopN1G6gNRaKTdRuoDUWik3UZoDUWkozRuoDUNv41o6Rod1rV15NshdulZ34Zrv/hDhfEUbZx83T8K2pxUp2ZzV6jhDTco2vwv1aZjH5TKR13Vh674avPD8zLdRMh6Zrv8Ax946vtL8SNHauVCtjArR8ZGPXPBsN7MmZ9uS1ehKmnoedSxE46zPG6KD1oryep7G6Eb7rfSvnz4qf8jNP9a+g2+6fpXz58Vf+Rmn+teXjfhPdy34jiT2pKX0pK8CJ9j0CiiigQUUUUAFFFFABRRRQAq8NSUUmPlx3oA+sf2IW23esnr+7b+QrsPhLK3/AAj3jYdR8/8ASvl74X/Fm/8Ahk9zJZAkzDa35VpeHPjhqWg2eqwwg7b8kv8Ajj/CgD334e2q3HwN11ZDgHIOf94V1Him4m+G3wd0668MaetxPLH88qrk96+UtL+NWp6X4TvdCj/1FyuCR65BrtPhx+1NqfhDR1069hOoWyDCo+CKAPpj9mzX/F/iTQdUuddtzFC8bMpIx2FeR/s865baP8db+K5lERmuGHXk1z19+2drP72OytRa20ilTGvAx+deDyeNL9fER1iCQx3O/eCDjBoA9g/aj+Get2vxKvr+KxkmtriTcjgZzXj2ueC9Z0G1jm1CwktonGVZhXumiftjX9vp8dvqmnLqDRjAdxn+tebfFn4zXXxQmQy2620K9EWgDd/Z5+Cl78Xri9htLryDDxlT9P8AGud+Mfw1uvhf4zuNNuJfOkjb7xqx8GvjdqXwfvJ5tPBYy9R/n6VjfFT4k33xN8Qy6veDbNIckUAaV/8AHDX7zw7ZaIsrQwW67VwcetetfCP9pfT49FXR/GEAvYIxtDPz/WvmAMVx3+tNxxjOAeTQB7f+0F8bLP4hyw6bpVsINOtuI1Hpz/jXiPI6nJ9aMnOT97tRQAcd+lfdX7HGvP4b+FWu3kIzNHDvB9+K+Fd23nGa9Q8C/G/U/BPhi70e2X9zcLtb/P4UAfVWj+LrL9prwjqGh6q6/wBrxqyop6k5ryn9nr4Xw6T8Ym0vXLb/AFcpVUkHXArwfwT8StT8EeJP7WspGSXfvIU9a7LVP2htX1HxbBr0K/Z7qNtzFepoA9v+KHxD8b6d8RzouhaW0VkkojQqnG2ut/ayjvG+C+n/AG4AXPkjcPQ815RN+2hdzWqs2lI98ox5zAZ+vWvPvih+0ZrfxM0P+zr3IjHSgD1f9iT5dN8R4PIQn9Fr5u+JbbvGF+ScuZmNbnwv+MF98M4b2OzUkXIw36f4Vxeuaq+salNfMAJJCTQB9V/sQ65CY9X0suoubiMqiHucCvKtT+FPiL/ha/2T+zpgTc5Bxwea898E+N9Q8D6zDqWnytFLGcnae9e9L+2jdm3UvpSNequBcEfNu9etAHT/ALZN5DpfhXQdEMqtcwxhXTPK8HivI/2U9SSz+LGmeY4RN+Mk+xrgfiJ8Q9R+ImtSahqEjMxbKgnpWPoOs3Hh/UYr60cxzxtuDKaAPbv2qPBesyeP727Swke2lbckmOMYrjvgB44l8D/ECxlErRRtKFfPA9K7S4/a01XUdJSxvbOO4Ozb5jDNefeA/h/rvxI8QLc6bYs2ZQzsvQc9aAPv3VPB+keAP7W8fLsxLH5qv7kCvgC6+L+rWfjm7121f995pIOa+pv2ovHh8J/Cu08KG63XflBJUB5FfC4wzFmAJPpQB7cv7WnjYdJ1B681Tvv2pvGWqWU1tJMjJINp69K8f4znbzSchQBwRxQB9pfDS4kn/Zt1uU4y8JOR9RXA/sayD/hZkh7lzXmmh/GrUtD8E3PhyIH7NKu0VlfDn4l3nw71htQtB+9Zs8UAdp8fNWl0j44ajdwnDxzjp+FfZPhHQNP+NnhfQPEFw6NLYoHcdemf8a+A9QutV+LnjQzRQtLe3T/cXrX2J4BGo/AH4JX8+ryGC5lhIihY89qAPA/2tvHC+JPHk1rbtmztH2RqD2AryzwjoN/eapYTw2ksqecoDKuRWb4h1mbXNblvZjvaRi3616b8O/2hZvh/o8VjHpkNwY/43HP86APcv2uNFvbzwL4bWG3eQrCNwUZx1r5e+Hun+LNL1pZ9DtZvt9ucsq8V7HfftparqMKx3GnxzqvRZBwB+dcJpf7Qd7pPjafXoLRE8xtxiA4oA92+Fvxy8S+IPEVroviTw/JcoxEbMyZx+tcF+194L0vQ/F1uumRIkly3MSjpkdKvf8NltHDvh0aOO7x/rdoz/OvC/G3xN1Pxt4kOrXrtJJv3qCelAH1D8B/hbb/DTwZ/wl2swD7S0fmRQuuTmvn/AOLmveIvH2uXWoXVtMtoCSq4OAtd5pf7YmsabocGmvZJNDEoUBuR/OsfxR+1Bc+JdJuLF9Jt0Eg2htvT9aAPMPAMix+LtMduFEy5J+tfTX7ZPhvUdaXQbyytZLm18sfvEGR0NfIsVxJFdi5iYpIG3DHQGvefDn7WGs6TocOnXlql8kS7VMnPFAHNfs/3T+FvitpMl+jQFZQG3jAHWvUf2xPh7qeoeNI9WsrSS6trohkdRkdK8G8ZfEO58U+Iv7XSJbOYNuHl8Yr1jwv+15qmmabDZ6jZLqAhGEaTk/zoA9e0S0k+Hv7Mc0WqL5E9xb4Eb8HrXln7Gcgb4kyOOhZj+lcB8V/j1q/xMxAxNtZjgQg8Yrnvhr8Sr34Z6y2oWYy/b8qAOh/aOmNx8S9S3DA3muH8H+FdT8UatHZaUrvdMcrs60zxj4quPGGsXF/ccPK2af4N8Y3vgvWoNRsv9fEcgmgD6w+BmoePfC2napbXsV15sAwqSdD0rV/aKjXWPggNT1m1jtta2bgpHOc1n/C39qGfVvD+oyahDCk0aZyerHivnL4qfGzWPiRePHdysLZWIEYPGKAPOhk8tg49KduHrUS9OgFLQBJuHrRuHrUdFAEm4etG4etR0UASbh60bh61HRQBJuHrRuHrUdFAEm4etG4etR0UASbh60bh61HRQBJuHrRuHrUdFAEm4etG4etR0UASbh60bh61HRQBJuHrRuHrUdFAEm4etG4etR0UASbh60bh61HRQBJuHrRuHrUdFAEm4etG4etR0UASbh60bh61HRQBJuHrRuHrUdFAEm4etG4etR0UASbh60bh61HRQBJuHrSbh2plFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFACmjtQaO1Jh0Q+0/wCPiL/eH86+o9D50u3/AN0V8uWn/HxF/vD+dfUXh/8A5B0PP8Ir3MCfOZtujVbt9KSjvmjNe3bQ+YTFopM0ZpCFopM0ZoAWikzRmgBaKTNGaAFopM0ZoAWikzRmgBaKTNGaAFopM0ZoAWikzRmgBaKTNGaAFopM0ZoAWikzRmgBaKTNGaAFopM0ZoAWikzRmgBaKTNGaAFopM0ZoAWikzRmgBaKTNGaAFru/hXMkevxszALjvXB1Ys76SxYNE5Rx3FVGTi9DOpTVSNjpfiJIJPEkjphl3dq7HVLhP8AhXMC+YCdn3a8kmupLiYvJIXY881O2pXBtVhaZmjAxt7VrKvKxz/VVy2Kx70LR+PFHpXLzXZ26WE/havnz4rf8jNP9a+gm+61fPvxW58TTfWvOxq909vLbc2hxH8I+lFK3GB7UlfPLqfYdAooooEFFFFABRRRQAUUUUAFFFFAB3oyR0pc8EdqSgAoyaMmigBeM+1IfaiigAooooAKXOetJRQAUUUUAFFFFABRmiigAooooAKKKKACiiigAooooAKM4oooAPUngd6734f/ABg1n4eQPFpshAcYBFcGPlzjv1pMnbjtQBu+LfHGq+NNUmvdSlaaaTkbjmsI/wCcUL8uMduKKACiiigAooooA3fCHii58I6tBfWLL58ZyHroPHPxn8QePolt9RuHeFfl254rg8mkHGffrQA/cSpJOB2ph60dgOwoJJoAP0pVbbSUueCO1ACGhd3ajPaigAopSSetJQAoO2kPPNFFABt3cZpMUtFAB0pOW60tKST1oASlXrikooAt21/NZxvHFKyKw5APWqTfMxOOTTjz160fWgAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAFNHakp6qCoo3DsLa/8AHxF/vD+dfUXh/wD5B0J/2RXy7FJsdXC4CHJrurX4r3ttCsag7VGM16mFqqnuePmGHlWs4nvhYbRyM0m73FeF/wDC3b70aj/hbl76NXp/XIWPC/s6qe5lh6ik3D1rw0fFy99DS/8AC3Lz0P50fXKY/wCzqp7juHrRuHrXh3/C3Lz0P50f8LcvPQ/nR9cph/Z9U9x3D1o3D1rw7/hbl56H86P+FuXnofzo+uUw/s+qe47h60bh614d/wALcvPQ/nR/wty89D+dH1ymH9n1T3HcPWjcPWvDv+FuXnofzo/4W5eeh/Oj65TD+z6p7juHrRuHrXh3/C3Lz0P50f8AC3Lz0P50fXKYf2fVPcdw9aNw9a8O/wCFuXnofzo/4W5eeh/Oj65TD+z6p7juHrRuHrXh3/C3Lz0P50f8LcvPQ/nR9cph/Z9U9x3D1o3D1rw7/hbl56H86P8Ahbl56H86PrlMP7PqnuO4etG4eteHf8LcvPQ/nR/wty89D+dH1ymH9n1T3HcPWjcPWvDv+FuXnofzo/4W5eeh/Oj65TD+z6p7juHrRuHrXh3/AAty89D+dH/C3Lz0P50fXKYf2fVPcdw9aNw9a8O/4W5eeh/Oj/hbl56H86PrlMP7PqnuO4etG4eteHf8LcvPQ/nR/wALcvPQ/nR9cph/Z9U9x3D1o3D1rw7/AIW5eeh/Oj/hbl56H86PrlMP7PqnuO4etG4eteHf8LcvPQ/nR/wty89D+dH1ymH9n1T3HcPWjcPWvDv+FuXnofzo/wCFuXnofzo+uUw/s+qe47h60u73rw3/AIW5eeh/Ok/4W5e+ho+uUw/s+qe6bqN1eF/8LdvfQ0f8LdvfQ0fXIB/Z1Q903Uu4eteFf8LevfQ0f8LevfRql4yA1ltU90465pd3vXhX/C3r30aj/hb176NS+uQK/s2rc903e4pQ3fIrwz/hb156Ghvi9en1FH1qF7i/s2dj3JmG08ivn74p/wDIzzf71X/+Ft3pU1x2va5Jrt+08v3ic1xYivGS0PSweDlSlqZTfeH0pKc2GY4ONtNryD6DoFFFFIAooopgFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAU7I49RTaKAFLHacdz0peAetNpeKbb6DvYduHrS+ZTeKOKnUfMuw7cD3o3D1phpKOVvqLmXYk3D1o3D1qOijl8w5l2JNw9aNw9ajoo5fMOZdiTcPWjcPWo6KOXzDmXYk3D1o3D1qOijl8w5l2JNw9aNw9ajoo5fMOZdiTcPWjcPWo6KOXzDmXYk3D1o3D1qOijl8w5l2JNw9aNw9ajoo5fMOZdiTcPWjcPWo6KOXzDmXYk3D1o3D1qOijl8w5l2JNw9aNw9ajoo5fMOZdiTcPWjcPWo6KOXzDmXYk3D1o3D1qOijl8w5l2JNw9aNw9ajoo5fMOZdiTcPWjcPWo6KOXzDmXYk3D1o3D1qOijl8w5l2JNw9aNwHeo6UUcrXUOZdh/mUeZTeKOKNR8y7DvM96TdTeKSnqHMuw/dS7qjoosHN5EmRRlfWo6KomyJQ49aTI9ajopMNBzbe3Wm0UUgCiiigAooooAKWkopfFsO11dC0lKeaSnbsStgooop2kOzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiii0gswooootILMKKKKLSCzCiiilyseoUUUU7SDUKKKKXKw1CiiijlYahRRRRysNQoooo5WGoUUUUcrDUKKKKOVhqFFFFHKw1CiiijlYahRRRRysNQoooo5WGoUUUUcrDUKKKKOVhqFFFFHKw1CiiijlYahRRRRysNQoooo5WGoUUUUcrDUKKKKOVhqFFFFO0g1Ciiii0g1Ciiii0g1CiiilysNQoooo5WGoUUUU7SFZhRRRRaQWYUUUUWkFmO2/LTWO3HGcnFFFEtIDWwqxlx94qOvFN8yiilTScdR2Vg30b6KKvlQrIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvo30UUcqCyDfRvooo5UFkG+jfRRRyoLIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvo30UUcqCyDfRvooo5UFkG+jfRRRyoLIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvo30UUcqCyDfRvooo5UFkG+jfRRRyoLIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvo30UUcqCyDfRvooo5UFkG+jfRRRyoLIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvo30UUcqCyDfRvooo5UFkG+jfRRRyoLIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvo30UUcqCyDfRvooo5UFkG+jfRRRyoLIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvo30UUcqCyDfRvooo5UFkG+jfRRRyoLIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvo30UUcqCyDfRvooo5UFkG+jfRRRyoLIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvo30UUcqCyDfRvooo5UFkG+jfRRRyoLIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvo30UUcqCyDfRvooo5UFkG+jfRRRyoLIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvo30UUcqCyDfRvooo5UFkG+jfRRRyoLIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvo30UUcqCyDfRvooo5UFkG+jfRRRyoLIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvo30UUcqCyDfRvooo5UFkG+jfRRRyoLIN9G+iijlQWQb6N9FFHKgsg30b6KKOVBZBvoooo5UFkf/Z)With this “naïve implementation”, the execution time increases linearly the bigger N is set to.

**Applying a 2000-year-old Formula:**

* **Carl Friedrich Gauss (1777 – 1855):**
  + Solves sum(n) as an elementary student.
  + Calculates 1 + 2 + 3 + … + 99 + 100 in seconds.
  + Triangle Numbers Formula:
    - 1 + 100 = 101 **1**
    - 2 + 99 = 101 **2**
    - 3 + 98 = 101 **3**
    - 4 + 97 = 101 **4**
    - … … 🡪 50 x 101 = **5050**
    - 47 + 54 = 101 **47**
    - 48 + 53 = 101 **48**
    - 49 + 52 = 101 **49**
    - 50 = 51 = 101 **50**
* **sum(N) = n \* (n + 1) /** 2 ; This formula likely predates Gauss and was likely known by the Pythagoreans in the 6th century BCE.
* Graphical user interface, application

  Description automatically generatedApplying this to a new function **sum\_optimized** gives us completely flat growth over time:

import time

def sum\_optimized(n: int) -> int:

    return n \* (n + 1) / 2

ranges = (10, 1000, 10000, 100000, 1000000)

print("\nMeasuring sum\_optimized: ")

for n in ranges:

    start = time.process\_time\_ns()

    sum\_optimized(n)

    end = time.process\_time\_ns()

    exec\_time = end - start

    print(f"sum\_optimized({n}) execution time: {exec\_time}ns")

**Section 16 – PART 4: The Unified Modeling Language (UML):**

**Introduction, UML Diagram Types:**

* **Unified Modeling Language:**
  + Graphical notation used to communicate the design of software systems.
* **Models:**
  + Functional: Use-Case Diagram
  + Structural: Class Diagram
  + Dynamic: Sequence Diagram
* **UML Usage:**
  + Sketch 🡪 Design and documentation
  + Reverse engineer
  + Create detailed blueprint
* **UML Diagrams:**
  + Standard graphical notation used to describe object-oriented systems.

**Class Diagrams:**

* Table

  Description automatically generated with medium confidenceHelpful in the early stages to plot out code.
* It’s quite common to leave out the attributes and operations in the early stages of development. We can add the details later as we realize what’s needed to fulfill the required functionality.

Diagram, table

Description automatically generated with medium confidenceGraphical user interface

Description automatically generated

* **Visibility:**
  + Controls who can access the attributes and the methods of our classes.
  + **+ public** – Can also be used by code outside of the object.
  + **- private** – Can only be accessed within the defining class.
  + Diagram

    Description automatically generated**# protected** – Accessible from defining and child classes.
* **Class Diagram:**
  + Provides an overview of the essential classes and shows the relationships between them.

**UML Relationships:**

* **Generalization:**
  + “B is an A”
  + One or more children
* **Association:**
  + Reference
  + Multiplicity
    - 0 – no instances
    - 0..1 – zero or exactly 1 instance
    - 1 – exactly one instance
    - 0..\* – zero or more instances
    - \* – zero or more instances
* **Navigability:**
  + One-way
* **Aggregation:**
  + “has-a”
* **Composition:**
  + “part-of”
* **Realization:**
  + Implement behavior
* **Dependency:**
  + Weak relations

Graphical user interface, diagram, application, Word

Description automatically generated

**UML Sequence Diagrams:**

* **Sequence Diagram:**
  + Describes the flow of logic in one particular scenario.
* Showed off a free UML modeling software called StarUML.io (<https://staruml.io/>) that he uses to model UMLs in this lecture.
  + Looks like I’m going to need more practice in the future in designing and reading UML sequence diagrams. Found a few YouTube videos about how to use StarUML.io, so those could be useful.
* Important: You don’t need to represent all the method calls in a sequence diagram.
* Focus on the important interactions between objects.

Diagram

Description automatically generated

**Activity Diagrams:**

* **Activity Diagram:**
  + Behavioral diagram used to describe workflows.

Diagram

Description automatically generatedDiagram

Description automatically generated

A picture containing diagram

Description automatically generated

Diagram

Description automatically generated

* **Trip Creation:**
  + Create new trip.
  + Conditional logic:
    - Type name
      * 🡪 [trip name not taken] 🡪 “Show New Trip Dialog”
      * 🡪 [else] 🡪 Prompt user

**Statechart Diagrams:**

* **Statechart Diagram:**
  + Describes the state changes of an object.

Diagram

Description automatically generated

Diagram

Description automatically generated

**Section 17 – PART 5: Careers in Programming – Web Development:**

**Overview:**

* **August 1991** – First website
* **1992** – 10 websites
* **1993** – 130 websites
* **2018** - **> 1,800,000,000 websites**
* **What you’ll learn in this section:**
  + How do websites work?
  + Core web technologies
  + Cons of web development
  + What’s next?

**How Do Websites Work?**

* **Web Development** is about creating applications that run in web browsers.
* Client sends request for a website, server responds (learned this a lot more in-depth in that Kali Linux Ethical Hacking course).
  + Server responds by sending several text files that will be compiled into a visual webpage: **HTML**, **CSS**, and **JavaScript**.
* **Uniform Resource Identifier (URI):**
  + A string of characters that identifies a particular resource.

**What’s HTML?** *“webpage-1.html”*

* **Hyper Text Markup Language (HTML):** 1989
  + Describes the structure and contents of web pages.
  + **<p>**This is a paragraph.**</p>**
  + **<h1>**Software Development from A to Z**</h1>** (h1 is like a chapter in a book)
  + **<h2>**Core Object-Orientation Concepts**</h2>** (h2 denotes sections in a chapter)
  + **<h3>**Brief History of Programming**</h3>** (h3 would be a subsection included in the section)

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

**Use CSS to Style Webpages:** *“webpage-1.html”*

* **Cascading Style Sheets (CSS):**
  + Provides the ability to style web pages.
* We first add a **<head></head>** element (below <html> and above <title>). This <head> element includes information that won’t be seen by the user.
* We then add a **<style></style>** element within <head>; we’re going to place the CSS definition inside this element. The following will apply the color “blue” to the “paragraph” element, **p**:

Text

Description automatically generated

* We can also change the font of elements:

Text

Description automatically generated

**JavaScript Basics:** *“webpage-1.html”*

* **JavaScript:**
  + A programming language that can be used to integrate dynamic features into websites.
* What can we do with JavaScript?
  + Change attributes dynamically
  + Display 3D graphics
  + Provide text editing features
  + Create eCommerce modules
  + Implement games
* We’re going to add a **<script>** tag near the bottom, just above the closing </body> tag.
  + The reason for this is that the web browser will load our code in the order that it’s written.
  + The JavaScript code operates on the HTML content, and therefore it’s important to load the HTML before the JavaScript.
* **JavaScript Data Types:**
  + String
  + Boolean
  + Number
  + Array
  + Object

**Implementing a Color Theme Picker in JavaScript:** *“webpage-1.html”*

* We’re going to implement interactive buttons for choosing Dark Theme or Light Theme.
* We started by adding **var firstHeading = document.querySelector(“h1”);** inside the **<script>** part.
  + The “**document**” portion refers to our HTML document that’s being loaded by the browser.
  + The **querySelector()** is a function returns the reference for a given element, in this case the first-level heading (**h1**).
  + Functions are building blocks that package functionality that you can reuse.
* We then change the **h1** heading color to read using:
  + **firstHeading.style.color = “red”;**
* We then repeated this process with the second-level heading, **h2**, and to the paragraph, **p**.

**JavaScript Functions and Event Handlers – Finishing the Color Theme Picker:** *“webpage-1.html”*

* This is where we’ll add the two buttons to switch between Light and Dark.
* We deleted all the commands to change the fonts red, but kept the variable callouts.
* Next we added a function with **function applyDarkTheme() {**, then we set all three variables to “white” inside of that and the background to “darkgray” (I decided to make mine “black” instead).
* We repeated this for **applyLightTheme()** but changed all the text to “blue” and the background to “white”.
* Outside of <script> but inside <body>, we added the code:
  + **<button onclick=”applyDarkTheme()”>Dark Theme</button>**
  + **<button onclick=”applyLightTheme()”>Light Theme</button>**
  + This code gave our webpage its buttons.

**Section 18 – PART 6: Ask Karoly – Student Q&A Videos:**

**Fixing Software Bugs:**

* “What are the methods for fixing bugs?” – student
  + There’s no standard approach for bug fixing.
* **Software Bug Types:**
  + Low-hanging fruit:
    - Fixing a typo.
  + Subtle issues:
    - May require a deeper knowledge of the software system.
  + Tip of the iceberg:
    - Bring deeper issues or critical design flaws to surface. These require a lot of time and analytical skills to get rid of them.
  + Sporadic:
    - Issues that pop up randomly and can’t be reproduced easily.
* **How are bugs detected?**
  + During development: Implementation, code reviews, or unit testing.
  + In the verification process: Testers.
  + During the productive usage: End users.
  + Once found, bugs are categorized and assigned to developers.
* **Bug Fixing Process:**
  + Reproduce & analyze
  + Correct
  + Validate solution
  + Release
* Not all bugs can be fixed.
  + Some can be caused by 3rd party applications or faulty system libraries or frameworks.
  + In such cases, we need to contact the given developer or company.

**Free Software for Agile Project Management:**

* “Would you suggest a free software for agile project management?” – student
* **JIRA** from Atlassian ([www.jira.com](http://www.jira.com))
  + Not free, but its self-hosted version is quite inexpensive for smaller teams.
  + SCRUM boards
  + Kanban mode
  + Reporting tools
  + Roadmap planning
  + Issue tracking
* **Asana** ([www.asana.com](http://www.asana.com))
  + His favorite, and free for teams up to 15 members.
  + Create projects & tasks
  + Assign
  + Add links, attach images
  + Conversations
* Asana also integrates a lot of useful third-party tools, like Slack, Dropbox, Harvest, etc.
* Premium account costs $6.25 per user per month.
* **Trello** ([www.trello.com](http://www.trello.com))
  + Create boards, lists & cards
  + Collaboration
  + Comments & attachments
  + Assign cards to members

**Section 19 – Conclusion:**

**Final Thoughts & Hints:**

* Free or inexpensive online resources.
* Join a Meetup group to find like-minded people.
* Learn by doing.
* Show off your work.