**INTRODUCTION TO SAKAM**



**Sakam Programming Language**

**Sakam Programming Language** is a simple language that you can use to write basic programs and even create simple 2D games. It’s not a very complex language. You can write programs or code in the Ol Chiki script, and the syntax is designed to be simple and human-readable, making it easy to understand.

### **Key Features:**

**Simple Programming**: Sakam is designed to be easy to understand. It allows you to write basic programs that can do simple tasks.

**2D Game Development**: With Sakam, you can create simple interactive 2D games, making it fun and creative.

**Ol Chiki Script**: You can write your code in the Ol Chiki script.

**Human-Readable Syntax**: The syntax of Sakam is simple and human-readable, allowing you to easily understand and write code without complicated rules.

**ᱚᱞ()**

ᱚᱞ() function in Sakam is a built-in utility that outputs information to the console. It is used to display text, numerical values, variables, or any other data type that can be represented as a string.

**Example1:**

ᱚᱞ (“ᱥᱟᱠᱟᱢ”)

**Output:**

ᱥᱟᱠᱟᱢ

**Example2:**

ᱚᱞ(“ᱥᱟᱠᱟᱢ\n” \* ᱕)

**Output:**

ᱥᱟᱠᱟᱢ

ᱥᱟᱠᱟᱢ

ᱥᱟᱠᱟᱢ

ᱥᱟᱠᱟᱢ

ᱥᱟᱠᱟᱢ

**Explanation**:

“ᱥᱟᱠᱟᱢ\n" is a string. The \n means a newline, so after printing, the next text will start on a new line.

\* ᱕ repeats the string “ᱥᱟᱠᱟᱢ\n" five times.

ᱚᱞ() displays the result.

**ᱡᱩᱫᱤ**

ᱡᱩᱫᱤ statement in Sakam is like a question in your code.

If the answer isᱥᱟᱹᱨᱤ, it does something. and If the answer is ᱮᱲᱮ, it does nothing

**Example1:**

ᱮᱞ = ᱑᱐

ᱡᱩᱫᱤ ᱮᱞ > ᱐:

ᱚᱞ("ᱮᱞ ᱫᱚ ᱯᱚᱡᱤᱴᱤᱵᱷ ᱜᱮᱭᱟ")

**Output:**

ᱮᱞ ᱫᱚ ᱯᱚᱡᱤᱴᱤᱵᱷ ᱜᱮᱭᱟ

**Example2:**

ᱮᱞ = ᱔

ᱡᱩᱫᱤ ᱮᱞ % ᱒ == ᱐:

ᱚᱞ("ᱮᱞ ᱫᱚ ᱥᱚᱢᱟᱱ ᱜᱮᱭᱟ")

**Output:**

ᱮᱞ ᱫᱚ ᱥᱚᱢᱟᱱ ᱜᱮᱭᱟ

**ᱵᱟᱝᱠᱷᱟᱱ**

ᱵᱟᱝᱠᱷᱟᱱ statement in Sakam is used when you want to do something if the ᱡᱩᱫᱤ condition is ᱮᱲᱮ (false).

**Example1:**

ᱮᱞ = -᱓

ᱡᱩᱫᱤ ᱮᱞ >= ᱐:

ᱚᱞ("ᱮᱞ ᱫᱚ ᱯᱚᱡᱤᱴᱤᱵᱷ ᱜᱮᱭᱟ")

ᱵᱟᱝᱠᱷᱟᱱ:

ᱚᱞ("ᱮᱞ ᱫᱚ ᱱᱮᱜᱮᱴᱤᱵᱷ ᱜᱮᱭᱟ")

**Output:**

ᱮᱞ ᱫᱚ ᱱᱮᱜᱮᱴᱤᱵᱷ ᱜᱮᱭᱟ

**Example2:**

ᱮᱞ = ᱗

ᱡᱩᱫᱤ ᱮᱞ % ᱒ == ᱐:

ᱚᱞ("ᱮᱞ ᱫᱚ ᱥᱚᱢᱟᱱ ᱜᱮᱭᱟ")

ᱵᱟᱝᱠᱷᱟᱱ:

ᱚᱞ("ᱮᱞ ᱫᱚ ᱵᱮᱜᱟᱨ ᱜᱮᱭᱟ")

**Output:**

ᱮᱞ ᱫᱚ ᱵᱮᱜᱟᱨ ᱜᱮᱭᱟ

**ᱳᱞᱡᱩᱫᱤ**

ᱳᱞᱡᱩᱫᱤ statement in Sakam stands for "ᱡᱩᱫᱤ ᱵᱟᱝᱠᱷᱟᱱ". It lets you check multiple conditions, one after another.

Sakam checks the conditions in order.

If one condition is ᱥᱟᱹᱨᱤ (true), it runs that block and skips the rest.

If none of the conditions are ᱥᱟᱹᱨᱤ (true), the else block (if present) runs.

**Example1:**

ᱮᱞ = ᱐

ᱡᱩᱫᱤ ᱮᱞ > ᱐:

ᱚᱞ("ᱮᱞ ᱫᱚ ᱯᱚᱡᱤᱴᱤᱵᱷ ᱜᱮᱭᱟ")

ᱳᱞᱡᱩᱫᱤ ᱮᱞ < ᱐:

ᱚᱞ("ᱮᱞ ᱫᱚ ᱱᱮᱜᱮᱴᱤᱵᱷ ᱜᱮᱭᱟ")

ᱵᱟᱝᱠᱷᱟᱱ:

ᱚᱞ("ᱮᱞ ᱫᱚ ᱥᱩᱱ ᱜᱮᱭᱟ")

**Output:**

ᱮᱞ ᱫᱚ ᱥᱩᱱ ᱜᱮᱭᱟ

**Example2:**

ᱴᱟᱲᱟᱝ = ᱑᱔

ᱡᱩᱫᱤ ᱴᱟᱲᱟᱝ < ᱑᱒:

ᱚᱞ("ᱥᱟᱹᱜᱩᱱ ᱥᱮᱛᱟᱜ")

ᱳᱞᱡᱩᱫᱤ ᱴᱟᱲᱟᱝ < ᱑᱘:

ᱚᱞ("ᱥᱟᱹᱜᱩᱱ ᱛᱟᱨᱟᱥᱤᱧ")

ᱵᱟᱝᱠᱷᱟᱱ:

ᱚᱞ("ᱥᱟᱹᱜᱩᱱ ᱥᱤᱸᱜᱟᱹᱲ")

**Output:**

ᱥᱟᱹᱜᱩᱱ ᱛᱟᱨᱟᱥᱤᱧ

**Example3:**

ᱩᱢᱮᱨ = ᱑᱐

ᱡᱩᱫᱤ ᱩᱢᱮᱨ < ᱑᱓:

ᱚᱞ("ᱟᱢ ᱫᱚ ᱢᱤᱫ ᱜᱤᱫᱽᱨᱟᱹ ᱠᱟᱱᱟᱢ")

ᱳᱞᱡᱩᱫᱤ ᱩᱢᱮᱨ < ᱒᱐:

ᱚᱞ("ᱟᱢ ᱫᱚ ᱢᱤᱫ ᱡᱩᱣᱟᱹᱱ ᱦᱚᱲ ᱠᱟᱱᱟᱢ")

ᱵᱟᱝᱠᱷᱟᱱ:

ᱚᱞ("ᱟᱢ ᱫᱚ ᱢᱤᱫ ᱵᱟᱹᱲᱛᱤ ᱦᱚᱲ ᱠᱟᱱᱟᱢ")

**Output:**

ᱟᱢ ᱫᱚ ᱢᱤᱫ ᱜᱤᱫᱽᱨᱟᱹ ᱠᱟᱱᱟᱢ

**Example4:**

ᱛᱚᱯᱢᱚᱱ = ᱓᱐

ᱡᱩᱫᱤ ᱛᱚᱯᱢᱚᱱ > ᱓᱕:

ᱚᱞ("ᱚᱫᱤ ᱜᱳᱨᱳᱢ ᱠᱟᱱᱟ")

ᱳᱞᱡᱩᱫᱤ ᱛᱚᱯᱢᱚᱱ > ᱒᱐:

ᱚᱞ("ᱞᱚᱞᱚ ᱜᱮᱭᱟ")

ᱳᱞᱡᱩᱫᱤ ᱛᱚᱯᱢᱚᱱ > ᱑᱐:

ᱚᱞ("ᱟᱹᱰᱤ ᱱᱟᱯᱟᱭᱟ")

ᱵᱟᱝᱠᱷᱟᱱ:

ᱚᱞ("ᱨᱮᱭᱟᱲ ᱜᱮᱭᱟ")

**Output:**

ᱞᱚᱞᱚ ᱜᱮᱭᱟ

**ᱞᱟᱹᱜᱤᱫ**

ᱞᱟᱹᱜᱤᱫ is used to repeat actions for every item in a group. It goes through each item one by one.

**Example1:**

ᱡᱚ\_ᱠᱚ = ["ᱥᱮᱵ", "ᱠᱟᱭᱮᱨᱟ", "ᱱᱮᱢᱩ"]

ᱞᱟᱹᱜᱤᱫ ᱡᱚ ᱨᱮ ᱡᱚ\_ᱠᱚ:

ᱚᱞ(ᱡᱚ)

**Output:**

ᱥᱮᱵ

ᱠᱟᱭᱮᱨᱟ

ᱱᱮᱢᱩ

**Example2:**

ᱞᱟᱹᱜᱤᱫ ᱮᱞ ᱨᱮ ᱥᱤᱢᱚ(᱑, ᱖):

ᱚᱞ(ᱮᱞ)

**Output:**

᱑

᱒

᱓

᱔

᱕

**ᱡᱚᱠᱷᱚᱱ**

ᱡᱚᱠᱷᱚᱱrepeats code as long as the condition is ᱥᱟᱹᱨᱤ (true) and stops when it becomes ᱮᱲᱮ(false).

**Example1:**

ᱞᱮᱠᱷᱟ = ᱑

ᱡᱚᱠᱷᱚᱱ ᱞᱮᱠᱷᱟ <= ᱕:

ᱚᱞ(ᱞᱮᱠᱷᱟ)

ᱞᱮᱠᱷᱟ += ᱑

**Output:**

᱑

᱒

᱓

᱔

᱕

**Example2:**

ᱮᱞ = ᱕

ᱡᱚᱠᱷᱚᱱ ᱮᱞ > ᱐:

ᱚᱞ(ᱮᱞ)

ᱮᱞ -= ᱑

**Output:**

᱕

᱔

᱓

᱒

᱑

**Example3:**

ᱞᱮᱠᱷᱟ = ᱐

ᱡᱚᱠᱷᱚᱱ ᱞᱮᱠᱷᱟ < ᱓:

ᱚᱞ("ᱥᱟᱠᱟᱢ")

ᱞᱮᱠᱷᱟ += ᱑

**Output:**

ᱥᱟᱠᱟᱢ

ᱥᱟᱠᱟᱢ

ᱥᱟᱠᱟᱢ

**Example4:**

ᱢᱩᱴ = ᱐

ᱡᱚᱠᱷᱚᱱ ᱢᱩᱴ < ᱑᱐:

ᱮᱞ = ᱒

ᱢᱩᱴ += ᱮᱞ

ᱚᱞ("ᱢᱩᱴ:", ᱢᱩᱴ)

**Output:**

ᱢᱩᱴ: ᱒

ᱢᱩᱴ: ᱔

ᱢᱩᱴ: ᱖

ᱢᱩᱴ: ᱘

ᱢᱩᱴ: ᱑᱐

**ᱨᱟᱹᱯᱩᱫ**

ᱨᱟᱹᱯᱩᱫis used to stop a loop immediately, even if the condition is still ᱥᱟᱹᱨᱤ (true).

**Example1:**

ᱞᱟᱹᱜᱤᱫ ᱮᱞ ᱨᱮ ᱥᱤᱢᱚ(᱑᱐):

ᱡᱩᱫᱤ ᱮᱞ == ᱗:

ᱨᱟᱹᱯᱩᱫ

ᱚᱞ(ᱮᱞ)

**Output:**

᱐

᱑

᱒

᱓

᱔

᱕

᱖

**Example2:**

ᱞᱮᱠᱷᱟ = ᱑

ᱡᱚᱠᱷᱚᱱ ᱥᱟᱹᱨᱤ:

ᱚᱞ(ᱞᱮᱠᱷᱟ)

ᱡᱩᱫᱤ ᱞᱮᱠᱷᱟ == ᱓:

ᱨᱟᱹᱯᱩᱫ

ᱞᱮᱠᱷᱟ += ᱑

**Output:**

᱑

᱒

᱓

**Example3:**

ᱢᱩᱴ = ᱐

ᱞᱟᱹᱜᱤᱫ ᱮᱞ ᱨᱮ ᱥᱤᱢᱚ(᱑, ᱑᱐):

ᱢᱩᱴ += ᱮᱞ

ᱡᱩᱫᱤ ᱢᱩᱴ > ᱑᱕:

ᱨᱟᱹᱯᱩᱫ

ᱚᱞ(ᱢᱩᱴ)

**Output:**

᱑

᱓

᱖

᱑᱐

᱑᱕

**Example4:**

ᱡᱚ\_ᱠᱚ = ["ᱥᱟᱵ", "ᱠᱟᱭᱨᱚ", "ᱱᱮᱢᱩ", "ᱛᱟᱢᱟᱛᱩᱞ"]

ᱞᱟᱹᱜᱤᱫ ᱡᱚ ᱨᱮ ᱡᱚ\_ᱠᱚ:

ᱡᱩᱫᱤ ᱡᱚ == "ᱱᱮᱢᱩ":

ᱨᱟᱹᱯᱩᱫ

ᱚᱞ(ᱡᱚ)

**Output:**

ᱥᱟᱵ

ᱠᱟᱭᱨᱚ

**ᱞᱟᱦᱟᱜ**

ᱞᱟᱦᱟᱜ is used to skip the rest of the code in the current loop and move to the next iteration. It doesn’t stop the loop; it just skips to the next step.

**Example1:**

ᱞᱟᱹᱜᱤᱫ ᱮᱞ ᱨᱮ ᱥᱤᱢᱚ(᱑, ᱖):

ᱡᱩᱫᱤ ᱮᱞ == ᱓:

ᱞᱟᱦᱟᱜ

ᱚᱞ(ᱮᱞ)

**Output:**

᱑

᱒

᱔

᱕

**Example2:**

ᱞᱟᱹᱜᱤᱫ ᱮᱞ ᱨᱮ ᱥᱤᱢᱚ(᱑, ᱖):

ᱡᱩᱫᱤ ᱮᱞ % ᱒ == ᱐:

ᱞᱟᱦᱟᱜ

ᱚᱞ(ᱮᱞ)

**Output:**

᱑

᱓

᱕

**Example3:**

ᱞᱮᱠᱷᱟ = ᱐

ᱡᱚᱠᱷᱚᱱ ᱞᱮᱠᱷᱟ < ᱕:

ᱞᱮᱠᱷᱟ += ᱑

ᱡᱩᱫᱤ ᱞᱮᱠᱷᱟ == ᱓:

ᱞᱟᱦᱟᱜ

ᱚᱞ(ᱞᱮᱠᱷᱟ)

**Output:**

᱑

᱒

᱔

᱕

**Example4:**

ᱥᱟᱵᱟᱫ\_ᱠᱩ = ["ᱦᱟᱭ", "ᱡᱚᱦᱟᱨ", "ᱛᱮᱦᱮᱧ", "ᱜᱮᱞ"]

ᱞᱟᱹᱜᱤᱫ ᱥᱟᱵᱟᱫ ᱨᱮ ᱥᱟᱵᱟᱫ\_ᱠᱩ:

ᱡᱩᱫᱤ ᱡᱮᱞᱮᱝ(ᱥᱟᱵᱟᱫ) < 4:

ᱞᱟᱦᱟᱜ

ᱚᱞ(ᱥᱟᱵᱟᱫ)

**Output:**

ᱡᱚᱦᱟᱨ

ᱛᱮᱦᱮᱧ

**ᱴᱷᱟᱹᱣᱠᱟᱹ**

ᱴᱷᱟᱹᱣᱠᱟᱹis used to create a **function**. A function is a block of reusable code that performs a specific task.

**Example1:**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱪᱮᱫ\_ᱭᱟ():

ᱚᱞ("ᱥᱟᱠᱟᱢ")

ᱪᱮᱫ\_ᱭᱟ()

**Output:**

ᱥᱟᱠᱟᱢ

**Example2:**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱪᱷᱟᱨ\_ᱠᱳᱱᱟ(ᱮᱞ):

ᱚᱞ(ᱮᱞ \* ᱮᱞ)

ᱪᱷᱟᱨ\_ᱠᱳᱱᱟ(6)

**Output:**

᱓᱖

**Example3:**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱥᱮᱞᱮᱫ\_ᱮᱞ(ᱚ, ᱵ):

ᱚᱞ(ᱚ + ᱵ)

ᱥᱮᱞᱮᱫ\_ᱮᱞ(᱓, ᱕)

**Output:**

᱘

**ᱨᱩᱣᱟᱹᱲ**

ᱨᱩᱣᱟᱹᱲ is used inside a function to send a value back to the place where the function was called.

**Example1:**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱢᱟᱨᱟᱝ\_ᱮᱞ(ᱚ, ᱵ):

ᱡᱩᱫᱤ ᱚ > ᱵ:

ᱨᱩᱣᱟᱹᱲ ᱚ

ᱵᱟᱝᱠᱷᱟᱱ:

ᱨᱩᱣᱟᱹᱲ ᱵ

ᱚᱞ(ᱢᱟᱨᱟᱝ\_ᱮᱞ(᱕, ᱘))

**Output:**

᱘

**Example2:**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱡᱮᱞᱮᱝ\_ᱥᱩᱛᱩᱢ(ᱥᱟᱵᱟᱫ):

ᱨᱩᱣᱟᱹᱲ ᱡᱮᱞᱮᱝ(ᱥᱟᱵᱟᱫ)

ᱚᱞ(ᱡᱮᱞᱮᱝ\_ᱥᱩᱛᱩᱢ("ᱥᱟᱠᱟᱢ"))

**Output:**

᱕

**Example3:**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱥᱚᱢ\_ᱛᱟᱞᱠᱟ(ᱮᱞᱠᱷᱟ):

ᱨᱩᱣᱟᱹᱲ ᱥᱮᱞᱮᱫ(ᱮᱞᱠᱷᱟ)

ᱚᱨᱡᱚ = ᱥᱚᱢ\_ᱛᱟᱞᱠᱟ([᱑, ᱒, ᱓, ᱔])

ᱚᱞ(ᱚᱨᱡᱚ)

**Output:**

᱑᱐

**Example4:**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱱᱮᱞᱢᱮ\_ᱤᱣᱮᱱ\_ᱚᱰ(ᱮᱞ):

ᱡᱩᱫᱤ ᱮᱞ % ᱒ == ᱐:

ᱨᱩᱣᱟᱹᱲ "ᱤᱣᱮᱱ"

ᱵᱟᱝᱠᱷᱟᱱ:

ᱨᱩᱣᱟᱹᱲ "ᱚᱰ"

ᱚᱨᱡᱚ = ᱱᱮᱞᱢᱮ\_ᱤᱣᱮᱱ\_ᱚᱰ(᱗)

ᱚᱞ(ᱚᱨᱡᱚ)

**Output:**

ᱚᱰ

**ᱟᱨ**

**ᱟᱨ** is a logical operator used to combine two conditions.

**Example1:**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱵᱩᱨᱩ(ᱮᱞ):

ᱡᱩᱫᱤ ᱮᱞ >= ᱑᱐ ᱟᱨ ᱮᱞ <= ᱒᱐:

ᱨᱩᱣᱟᱹᱲ "ᱮᱞ ᱫᱚ ᱯᱟᱥᱱᱟᱣ ᱨᱮ ᱢᱮᱱᱟᱜᱼᱟ"

ᱵᱟᱝᱠᱷᱟᱱ:

ᱨᱩᱣᱟᱹᱲ "ᱮᱞ ᱫᱚ ᱨᱮᱸᱡᱽ ᱵᱟᱦᱨᱮ ᱨᱮ ᱢᱮᱱᱟᱜᱼᱟ"

ᱚᱞ(ᱵᱩᱨᱩ(᱑᱑))

**Output:**

ᱮᱞ ᱫᱚ ᱯᱟᱥᱱᱟᱣ ᱨᱮ ᱢᱮᱱᱟᱜᱼᱟ

**Example2:**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱞᱚᱜᱤᱱ(ᱵᱮᱵᱷᱟᱨᱤᱭᱟᱹ\_ᱧᱩᱛᱩᱢ, ᱯᱟᱥᱣᱟᱨᱰ):

ᱡᱩᱫᱤ ᱵᱮᱵᱷᱟᱨᱤᱭᱟᱹ\_ᱧᱩᱛᱩᱢ == "ᱢᱟᱞᱤᱠ" ᱟᱨ ᱯᱟᱥᱣᱟᱨᱰ == "᱑᱒᱓᱔᱕":

ᱨᱩᱣᱟᱹᱲ "ᱞᱚᱜᱤᱱ ᱡᱤᱛᱠᱟᱹᱨ"

ᱵᱟᱝᱠᱷᱟᱱ:

ᱨᱩᱣᱟᱹᱲ "ᱚᱠᱟᱹᱡᱽᱣᱟᱹ"

ᱚᱞ(ᱞᱚᱜᱤᱱ("ᱢᱟᱞᱤᱠ", "᱑᱒᱓᱔᱕"))

ᱚᱞ(ᱞᱚᱜᱤᱱ("ᱷᱳᱨ", "᱑᱒᱓᱔᱕"))

**Output:**

ᱞᱚᱜᱤᱱ ᱡᱤᱛᱠᱟᱹᱨ

ᱚᱠᱟᱹᱡᱽᱣᱟ

**Example2:**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱱᱮᱞ\_ᱯᱟᱨᱳᱢ(ᱮᱞᱠᱷᱟ, ᱵᱤᱜᱭᱚᱱ):

ᱡᱩᱫᱤ ᱮᱞᱠᱷᱟ >= ᱔᱐ and ᱵᱤᱜᱭᱚᱱ >= ᱔᱐:

ᱨᱩᱣᱟᱹᱲ "ᱟᱢᱮᱢ ᱯᱟᱥ ᱮᱱᱟ"

ᱵᱟᱝᱠᱷᱟᱱ:

ᱨᱩᱣᱟᱹᱲ "ᱟᱢᱮᱢ ᱦᱟᱨᱟᱣ ᱮᱱᱟ"

ᱚᱞ(ᱱᱮᱞ\_ᱯᱟᱨᱳᱢ(᱔᱕, ᱕᱐))

ᱚᱞ(ᱱᱮᱞ\_ᱯᱟᱨᱳᱢ(᱓᱐, ᱕᱐))

**Output:**

ᱟᱢᱮᱢ ᱯᱟᱥ ᱮᱱᱟ

ᱟᱢᱮᱢ ᱦᱟᱨᱟᱣ ᱮᱱᱟ

**Example3:**ᱴᱷᱟᱹᱣᱠᱟᱹ ᱜᱟᱫᱤ\_ᱪᱷᱟᱞᱳᱣ(ᱩᱢᱮᱨ, ᱞᱟᱭᱥᱮᱱᱥ):

ᱡᱩᱫᱤ ᱩᱢᱮᱨ >= 18 ᱟᱨ ᱞᱟᱭᱥᱮᱱᱥ:

ᱨᱩᱣᱟᱹᱲ "ᱜᱟᱫᱤ ᱪᱷᱟᱞᱳᱣ ᱫᱟᱨᱤ ᱚᱷ"

ᱵᱟᱝᱠᱷᱟᱱ:

ᱨᱩᱣᱟᱹᱲ "ᱜᱟᱫᱤ ᱵᱟᱝ ᱪᱷᱟᱞᱳᱣ ᱫᱟᱨᱤ ᱚᱷ"

ᱚᱞ(ᱜᱟᱫᱤ\_ᱪᱷᱟᱞᱳᱣ(20, ᱥᱟᱹᱨᱤ))

ᱚᱞ(ᱜᱟᱫᱤ\_ᱪᱷᱟᱞᱳᱣ(16, ᱥᱟᱹᱨᱤ))

**Output:**

ᱜᱟᱫᱤ ᱪᱷᱟᱞᱳᱣ ᱫᱟᱨᱤ ᱚᱷ

ᱜᱟᱫᱤ ᱵᱟᱝ ᱪᱷᱟᱞᱳᱣ ᱫᱟᱨᱤ ᱚᱷ

**ᱟᱨ\_ᱵᱟᱝ**

ᱟᱨ\_ᱵᱟᱝ is a logical operator used to check if at least one condition is ᱥᱟᱹᱨᱤ (true)

**Example1:**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱪᱷᱩᱴ(ᱩᱢᱮᱨ, ᱯᱟᱹᱴᱷᱩᱣᱟᱹ):

ᱡᱩᱫᱤ ᱩᱢᱮᱨ < ᱑᱘ ᱟᱨ\_ᱵᱟᱝ ᱯᱟᱹᱴᱷᱩᱣᱟᱹ:

ᱨᱩᱣᱟᱹᱲ "ᱪᱷᱩᱴ ᱮᱢ ᱱᱚᱢ ᱟᱷ"

ᱵᱟᱝᱠᱷᱟᱱ:

ᱨᱩᱣᱟᱹᱲ "ᱪᱷᱩᱴ ᱵᱟᱝ ᱱᱚᱢ ᱟᱷ"

ᱚᱞ(ᱪᱷᱩᱴ(᱑᱖, ᱮᱲᱮ))

ᱚᱞ(ᱪᱷᱩᱴ(᱒᱕, ᱥᱟᱹᱨᱤ))

ᱚᱞ(ᱪᱷᱩᱴ(᱓᱐, ᱮᱲᱮ))

**Output:**

ᱪᱷᱩᱴ ᱮᱢ ᱱᱚᱢ ᱟᱷ

ᱪᱷᱩᱴ ᱮᱢ ᱱᱚᱢ ᱟᱷ

ᱪᱷᱩᱴ ᱵᱟᱝ ᱱᱚᱢ ᱟᱷ

.

**ᱵᱟᱝ**

ᱵᱟᱝ operator in Sakam is used to reverse a condition.

**Example1:**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱮᱞ\_ᱱᱮᱞᱢᱮ(ᱮᱞ):

ᱡᱩᱫᱤ ᱵᱟᱝ ᱮᱞ > ᱐:

ᱨᱩᱣᱟᱹᱲ "ᱵᱟᱝ ᱯᱚᱡᱤᱴᱤᱵᱷ"

ᱵᱟᱝᱠᱷᱟᱱ:

ᱨᱩᱣᱟᱹᱲ "ᱯᱚᱡᱤᱴᱤᱵᱷ"

ᱚᱞ(ᱮᱞ\_ᱱᱮᱞᱢᱮ(-᱕))

ᱚᱞ(ᱮᱞ\_ᱱᱮᱞᱢᱮ(᱓))

**Output:**

ᱵᱟᱝ ᱯᱚᱡᱤᱴᱤᱵᱷ

ᱯᱚᱡᱤᱴᱤᱵᱷ

**Example2:**

ᱮᱠᱴᱤᱵᱷ = ᱥᱟᱹᱨᱤ

ᱮᱠᱴᱤᱵᱷ = ᱵᱟᱝ ᱮᱠᱴᱤᱵᱷ

ᱚᱞ(ᱮᱠᱴᱤᱵᱷ)

**Output:**

ᱮᱲᱮ

**ᱫᱚ**

ᱫᱚoperator is used to check if two objects are the same, meaning they refer to the exact same memory location. It checks identity, not equality.

**Example1:**

ᱟ = [᱑, ᱒, ᱓]

ᱵ = ᱟ

ᱚᱞ(ᱟ ᱫᱚ ᱵ)

**Output:**

ᱥᱟᱹᱨᱤ # because ᱟ and ᱵ point to the same list object.

**Example2:**

ᱟ = [᱑, ᱒, ᱓]

ᱵ = [᱑, ᱒, ᱓]

ᱚᱞ(ᱟ ᱫᱚ ᱵ)

**Output:**

ᱮᱲᱮ # because ᱟ and ᱵ are two different list objects, even though they have the same value.

**Example3:**

ᱟ = "ᱥᱟᱠᱟᱢ"

ᱵ = "ᱥᱟᱠᱟᱢ"

ᱚᱞ(ᱟ ᱫᱚ ᱵ)

**Output:**

ᱥᱟᱹᱨᱤ # because small strings are cached and point to the same object.

**ᱨᱮ**

ᱨᱮ is used to check if a value is present in a sequence. It returns ᱥᱟᱹᱨᱤ if the value exists in the sequence, otherwise it returns ᱮᱲᱮ.

**Example1:**

ᱡᱚ\_ᱠᱚ = ["ᱥᱮᱵ", "ᱠᱟᱭᱨᱟ", "ᱞᱤᱪᱷᱩ"]

ᱚᱞ("ᱥᱮᱵ" ᱨᱮ ᱡᱚ\_ᱠᱚ)

ᱚᱞ("ᱟᱹᱝᱜᱩᱨ" ᱨᱮ ᱡᱚ\_ᱠᱚ)

**Output:**

ᱥᱟᱹᱨᱤ

ᱮᱲᱮ

**Example2:**

ᱮᱞ = [᱑, ᱒, ᱓, ᱔, ᱕]

ᱚᱞ(᱗ ᱵᱟᱝ ᱨᱮ ᱮᱞ)

ᱚᱞ(᱓ ᱵᱟᱝ ᱨᱮ ᱮᱞ)

**Output:**

ᱥᱟᱹᱨᱤ

ᱮᱲᱮ

**Example3:**

ᱨᱚᱝ\_ᱠᱚ = ["ᱟᱨᱟᱜ", "ᱞᱤᱞ", "ᱦᱟᱹᱨᱭᱟᱹᱲ"]

ᱞᱟᱹᱜᱤᱫ ᱨᱚᱝ ᱨᱮ ᱨᱚᱝ\_ᱠᱚ:

ᱡᱩᱫᱤ "ᱤ" ᱨᱮ ᱨᱚᱝ:

ᱚᱞ(ᱨᱚᱝ)

**Output:**

ᱞᱤᱞ

**ᱪᱟᱱᱟᱪ**

ᱪᱟᱱᱟᱪis like a recipe, and an **object** is what you create using that recipe. It combines related data and actions into one structure.

**Example1:**

ᱪᱟᱱᱟᱪ ᱥᱮᱛᱟ:

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱵᱟᱨᱠ(ᱱᱤᱡᱮᱨ):

ᱚᱞ("ᱣᱩᱯ!")

ᱤᱧᱤᱡ\_ᱥᱮᱛᱟ = ᱥᱮᱛᱟ()

ᱤᱧᱤᱡ\_ᱥᱮᱛᱟ.ᱵᱟᱨᱠ()

**Output:**

ᱣᱩᱯ!

**Example2:**

ᱪᱟᱱᱟᱪ ᱥᱮᱛᱟ:

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱠᱚᱨᱮᱱ(ᱱᱤᱡᱮᱨ, ᱧᱩᱛᱩᱢ):

ᱱᱤᱡᱮᱨ.ᱧᱩᱛᱩᱢ = ᱧᱩᱛᱩᱢ

ᱤᱧᱤᱡ\_ᱥᱮᱛᱟ = ᱥᱮᱛᱟ("ᱥᱟᱠᱟᱢ")

ᱚᱞ(ᱤᱧᱤᱡ\_ᱥᱮᱛᱟ.ᱧᱩᱛᱩᱢ)

**Output:**

ᱥᱟᱠᱟᱢ

**Example3:**

ᱪᱟᱱᱟᱪ ᱥᱮᱛᱟ:

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱠᱚᱨᱮᱱ(ᱱᱤᱡᱮᱨ, ᱧᱩᱛᱩᱢ="ᱵᱟᱝ ᱵᱟᱰᱟᱭᱚᱜ ᱠᱟᱱᱟ"):

ᱱᱤᱡᱮᱨ.ᱧᱩᱛᱩᱢ = ᱧᱩᱛᱩᱢ

ᱥᱮᱛᱟ1 = ᱥᱮᱛᱟ("ᱠᱳᱞ")

ᱥᱮᱛᱟ2 = ᱥᱮᱛᱟ()

ᱚᱞ(ᱥᱮᱛᱟ1.ᱧᱩᱛᱩᱢ)

ᱚᱞ(ᱥᱮᱛᱟ2.ᱧᱩᱛᱩᱢ)

**Output:**

ᱠᱳᱞ

ᱵᱟᱝ ᱵᱟᱰᱟᱭᱚᱜ ᱠᱟᱱᱟ

**Example4:**

ᱪᱟᱱᱟᱪ ᱷᱤᱥᱟᱵ:

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱥᱮᱞᱮᱫ\_ᱢᱮ(ᱱᱤᱡᱮᱨ, ᱟ, ᱵ):

ᱨᱩᱣᱟᱹᱲ ᱟ + ᱵ

ᱷᱤᱥ = ᱷᱤᱥᱟᱵ()

ᱚᱨᱡᱚ = ᱷᱤᱥ.ᱥᱮᱞᱮᱫ\_ᱢᱮ(᱕, ᱓)

ᱚᱞ(ᱚᱨᱡᱚ)

**Output:**

᱘

**More Code**

1. **CODE**

ᱮᱞᱠᱩ = [᱑, ᱒, ᱓, ᱔, ᱕, ᱖, ᱗, ᱘, ᱙, ᱑᱐]

ᱚᱨᱡᱚ = [ᱽ\*\*᱒ ᱡᱩᱫᱤ ᱽ % ᱒ == 0 ᱵᱟᱝᱠᱷᱟᱱ ᱽ\*\*᱓ ᱞᱟᱹᱜᱤᱫ ᱽ ᱨᱮ ᱮᱞᱠᱩ ᱡᱩᱫᱤ ᱽ > ᱓]

ᱚᱞ(ᱚᱨᱡᱚ)

1. **CODE**

ᱽ = ᱑᱐

ᱚᱵᱚᱥᱛᱟ = "ᱮᱶᱮᱱ" ᱡᱩᱫᱤ ᱽ % ᱒ == ᱐ ᱵᱟᱝᱠᱷᱟᱱ "ᱳᱫ"

ᱚᱞ(f"ᱮᱞ {ᱽ} ᱫᱚ {ᱚᱵᱚᱥᱛᱟ} ᱠᱟᱱᱟ.")

1. **CODE**

ᱟ, ᱵ, ᱪ = ᱕, ᱑0, ᱑᱕

ᱡᱩᱫᱤ ᱟ < ᱵ < ᱪ:

ᱚᱞ("ᱮᱞ ᱫᱳ ᱠᱟᱛᱤᱪ ᱠᱷᱳᱩᱮ ᱞᱟᱛᱩ ᱜᱮᱭᱚ")

ᱳᱞᱡᱩᱫᱤ ᱟ > ᱵ > ᱪ:

ᱚᱞ("ᱮᱞ ᱫᱳ ᱞᱟᱛᱩ ᱠᱷᱳᱩᱮ ᱠᱟᱛᱤᱪ ᱜᱮᱭᱚ")

ᱳᱞᱡᱩᱫᱤ ᱟ == ᱵ == ᱪ:

ᱚᱞ("ᱮᱞ ᱫᱳ ᱥᱚᱢᱟᱱ ᱜᱮᱭᱟ")

ᱵᱟᱝᱠᱷᱟᱱ:

ᱚᱞ("ᱵᱟᱝ ᱠᱟᱛᱤᱪ ᱟᱷ ᱵᱟᱝ ᱠᱟᱛᱤᱪ ᱟᱷ ᱵᱟᱝ ᱥᱚᱢᱟᱱ ᱟᱷ")

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱯᱷᱟᱭᱵᱚᱱᱟᱪᱤ(ᱱ):

ᱟ, ᱵ = ᱐, ᱑

ᱞᱟᱹᱜᱤᱫ \_ ᱨᱮ ᱥᱤᱢᱚ(ᱱ):

ᱚᱯᱥᱚᱨ ᱟ

ᱟ, ᱵ = ᱵ, ᱟ + ᱵ

ᱞᱟᱹᱜᱤᱫ ᱮᱞ ᱨᱮ ᱯᱷᱟᱭᱵᱚᱱᱟᱪᱤ(᱑᱐):

ᱚᱞ(ᱮᱞ)

1. **CODE**

ᱮᱞ\_ᱠᱩ = [᱑, -᱒, ᱓, -᱔, ᱕]

ᱞᱟᱹᱜᱤᱫ ᱮᱞ ᱨᱮ ᱮᱞ\_ᱠᱩ:

ᱡᱩᱫᱤ ᱮᱞ < ᱐:

ᱞᱟᱦᱟᱜ

ᱚᱞ(ᱮᱞ)

1. **CODE**

ᱽ = ᱑᱕

ᱚᱨᱡᱚ = (

"ᱠᱟᱛᱤᱪ" ᱡᱩᱫᱤ ᱽ < ᱑᱐ ᱵᱟᱝᱠᱷᱟᱱ

"ᱛᱟᱞᱟ" ᱡᱩᱫᱤ ᱽ < ᱒᱐ ᱵᱟᱝᱠᱷᱟᱱ

"ᱢᱟᱨᱟᱝ"

)

ᱚᱞ(ᱚᱨᱡᱚ)

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱡᱟᱱᱚᱢ\_ᱪᱷᱟᱨᱠᱩᱱᱟ(ᱥᱤᱢᱟᱹ):

ᱞᱟᱹᱜᱤᱫ ᱽ ᱨᱮ ᱥᱤᱢᱚ(ᱥᱤᱢᱟᱹ):

ᱚᱯᱥᱚᱨ ᱽ\*\*᱒

ᱞᱟᱹᱜᱤᱫ ᱪᱷᱟᱨᱠᱩᱱᱟ ᱨᱮ ᱡᱟᱱᱚᱢ\_ᱪᱷᱟᱨᱠᱩᱱᱟ(5):

ᱚᱞ(ᱪᱷᱟᱨᱠᱩᱱᱟ)

1. **CODE**

ᱰᱟᱴᱟ = {"ᱚ": ᱑, "ᱛ": ᱒, "ᱜ": ᱓}

ᱪᱟᱵᱷᱤ = "ᱛ"

ᱜᱚᱱᱚᱝ = ᱰᱟᱴᱟ[ᱪᱟᱵᱷᱤ] ᱡᱩᱫᱤ ᱪᱟᱵᱷᱤ ᱨᱮ ᱰᱟᱴᱟ ᱵᱟᱝᱠᱷᱟᱱ "ᱵᱟᱝ ᱧᱟᱢ ᱮᱱᱟ"

ᱚᱞ(ᱜᱚᱱᱚᱝ)

1. **CODE**

ᱰᱟᱴᱟ = [᱑, -᱑, ᱒, -᱒, ᱓]

ᱵᱟᱪᱷᱳᱣ\_ᱰᱟᱴᱟ = [ᱽ ᱞᱟᱹᱜᱤᱫ ᱽ ᱨᱮ ᱰᱟᱴᱟ ᱡᱩᱫᱤ ᱽ > ᱐]

ᱚᱞ(ᱵᱟᱪᱷᱳᱣ\_ᱰᱟᱴᱟ)

1. **CODE**

ᱮᱞ\_ᱠᱩ = [᱑, ᱒, ᱓, ᱔]

ᱢᱩᱴ = ᱐

ᱤᱫᱽ = ᱐

ᱡᱚᱠᱷᱚᱱ ᱤᱫᱽ < ᱡᱮᱞᱮᱝ(ᱮᱞ\_ᱠᱩ):

ᱢᱩᱴ += ᱮᱞ\_ᱠᱩ[ᱤᱫᱽ]

ᱤᱫᱽ += ᱑

ᱚᱞ("ᱢᱩᱴ:", ᱢᱩᱴ)

1. **CODE**

ᱚᱨᱡᱚ = [ ]

ᱞᱟᱹᱜᱤᱫ ᱽ ᱨᱮ ᱥᱤᱢᱚ(᱕):

ᱚᱨᱡᱚ.ᱡᱳᱨᱟᱳᱢᱮ(ᱽ \* ᱒)

ᱚᱞ("ᱚᱨᱡᱚ:", ᱚᱨᱡᱚ)

1. **CODE**

ᱞᱟᱹᱜᱤᱫ ᱤ ᱨᱮ ᱥᱤᱢᱚ(᱑᱐):

ᱞᱟᱹᱜᱤᱫ ᱡ ᱨᱮ ᱥᱤᱢᱚ(᱑᱐):

ᱡᱩᱫᱤ ᱤ \* ᱡ > ᱕᱐:

ᱚᱞ(ᱤ, ᱡ)

ᱨᱟᱹᱯᱩᱫ

ᱡᱩᱫᱤ ᱤ \* ᱡ > ᱕᱐:

ᱨᱟᱹᱯᱩᱫ

1. **CODE**

ᱢᱮᱴᱨᱤᱠᱥ = [[᱑, ᱒, ᱓], [᱔, ᱕, ᱖], [᱗, ᱘, ᱙]]

ᱡᱚᱥ = ᱕

ᱞᱟᱹᱜᱤᱫ ᱛᱷᱟᱨ ᱨᱮ ᱢᱮᱴᱨᱤᱠᱥ:

ᱡᱩᱫᱤ ᱡᱚᱥ ᱨᱮ ᱛᱷᱟᱨ:

ᱚᱞ("ᱱᱨᱭᱟ ᱨᱮ ᱢᱮᱱᱟ ᱟᱷ:", ᱛᱷᱟᱨ)

ᱨᱟᱹᱯᱩᱫ

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱷᱤᱥᱟᱵ\_ᱮᱨᱤᱭᱟ(ᱛᱨᱤᱭᱥᱟ):

ᱡᱩᱫᱤ ᱛᱨᱤᱭᱥᱟ < ᱐:

ᱨᱩᱣᱟᱹᱲ ᱚᱠᱚᱭ\_ᱦᱚᱸ\_ᱵᱟᱝ

ᱨᱩᱣᱟᱹᱲ ᱓.᱑᱔ \* ᱛᱨᱤᱭᱥᱟ \* ᱛᱨᱤᱭᱥᱟ

ᱚᱨᱡᱚ = ᱷᱤᱥᱟᱵ\_ᱮᱨᱤᱭᱟ(᱕)

ᱡᱩᱫᱤ ᱚᱨᱡᱚ:

ᱚᱞ("ᱮᱨᱤᱭᱟ ᱫᱚ", ᱚᱨᱡᱚ)

1. **CODE**

ᱮᱞ\_ᱠᱩ = ᱥᱤᱢᱚ(᱑᱐, ᱑᱖)

ᱞᱟᱹᱜᱤᱫ ᱱ ᱨᱮ ᱮᱞ\_ᱠᱩ:

ᱚᱞ("ᱮᱞ:", ᱱ)

1. **CODE**

ᱥᱤᱢᱟᱹ = ᱑᱐᱐

ᱱᱤᱛᱚᱜ = ᱑

ᱡᱚᱠᱷᱚᱱ ᱱᱤᱛᱚᱜ < ᱥᱤᱢᱟᱹ:

ᱱᱤᱛᱚᱜ \*= ᱒

ᱚᱞ("ᱱᱤᱛᱚᱜ ᱜᱚᱱᱚᱝ:", ᱱᱤᱛᱚᱜ)

1. **CODE**

ᱞᱟᱹᱜᱤᱫ ᱤ ᱨᱮ ᱥᱤᱢᱚ(᱑᱐):

ᱞᱟᱹᱜᱤᱫ ᱡ ᱨᱮ ᱥᱤᱢᱚ(᱑᱐):

ᱡᱩᱫᱤ ᱤ + ᱡ > ᱑᱕:

ᱚᱞ("ᱨᱮ ᱛᱤᱸᱜᱩᱱ ᱠᱟᱱᱟ ᱤ =", ᱤ, "ᱡ =", ᱡ)

ᱨᱟᱹᱯᱩᱫ

ᱵᱟᱝᱠᱷᱟᱱ:

ᱞᱟᱦᱟᱜ

ᱨᱟᱹᱯᱩᱫ

1. **CODE**

ᱞᱟᱹᱜᱤᱫ ᱽ ᱨᱮ ᱥᱤᱢᱚ(᱓):

ᱞᱟᱹᱜᱤᱫ ᱭ ᱨᱮ ᱥᱤᱢᱚ(᱓):

ᱡᱩᱫᱤ ᱽ == ᱭ:

ᱞᱟᱦᱟᱜ

ᱚᱞ("ᱡᱚᱲ:", ᱽ, ᱭ)

1. **CODE**

ᱢᱟᱨᱥᱟᱞ = "ᱥᱟᱥᱟᱝ"

ᱡᱩᱫᱤ ᱢᱟᱨᱥᱟᱞ == "ᱦᱟᱹᱨᱭᱟᱹᱲ":

ᱚᱞ("ᱥᱮᱱᱚᱜ ᱢᱮ")

ᱳᱞᱡᱩᱫᱤ ᱢᱟᱨᱥᱟᱞ == "ᱥᱟᱥᱟᱝ":

ᱚᱞ("ᱵᱟᱹᱭ ᱵᱟᱹᱭ ᱛᱮ ᱪᱟᱞᱟᱜ ᱢᱮ")

ᱳᱞᱡᱩᱫᱤ ᱢᱟᱨᱥᱟᱞ == "ᱟᱨᱟᱜ":

ᱚᱞ("ᱛᱤᱱᱜᱩ ᱢᱮ")

1. **CODE**

ᱜᱚᱱᱚᱝ = ᱑

ᱡᱚᱠᱷᱚᱱ ᱜᱚᱱᱚᱝ < ᱑᱐᱐᱐:

ᱜᱚᱱᱚᱝ \*= ᱒

ᱚᱞ("ᱜᱚᱱᱚᱝ ᱫᱳ", ᱜᱚᱱᱚᱝ)

1. **CODE**

ᱮᱞ\_ᱠᱩ = [᱑᱒, ᱑᱕, ᱑᱘, ᱒᱑, ᱒᱔]

ᱵᱟᱪᱷᱳᱣ = [ᱽ ᱞᱟᱹᱜᱤᱫ ᱽ ᱨᱮ ᱮᱞ\_ᱠᱩ ᱡᱩᱫᱤ ᱽ % ᱓ == ᱐ ᱟᱨ ᱽ > ᱑᱕]

ᱡᱩᱫᱤ ᱵᱟᱪᱷᱳᱣ:

ᱚᱞ(ᱵᱟᱪᱷᱳᱣ)

1. **CODE**

ᱽ = ᱐

ᱞᱟᱹᱜᱤᱫ ᱮᱞ ᱨᱮ ᱥᱤᱢᱚ(᱑᱐):

ᱡᱩᱫᱤ ᱮᱞ % ᱓ == ᱐:

ᱞᱟᱦᱟᱜ

ᱽ += ᱑

ᱚᱞ("ᱯᱨᱚᱥᱮᱥ ᱦᱩᱭ ᱟᱠᱟᱱᱟ:", ᱮᱞ)

ᱚᱞ("ᱢᱩᱴ ᱯᱨᱚᱥᱮᱥ ᱦᱩᱭ ᱟᱠᱟᱱᱟ:", ᱽ)

1. **CODE**

ᱵᱮᱞᱮᱱᱥ = ᱑᱐᱐

ᱫᱚᱨ = ᱐.᱐᱕

ᱥᱮᱨᱢᱟ = ᱐

ᱡᱚᱠᱷᱚᱱ ᱵᱮᱞᱮᱱᱥ < ᱒᱐᱐:

ᱵᱮᱞᱮᱱᱥ \*= ᱑ + ᱫᱚᱨ

ᱥᱮᱨᱢᱟ += ᱑

ᱚᱞ(ᱥᱮᱨᱢᱟ,"ᱥᱮᱨᱢᱟ ᱞᱟᱜᱳᱣ ᱱᱟ ᱫᱚᱨ ᱫᱷᱤᱨ ᱨᱮ")

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱯᱷᱤᱵᱚᱱᱟᱪᱤ(ᱱ):

ᱟ, ᱵ = ᱐, ᱑

ᱞᱟᱹᱜᱤᱫ \_ ᱨᱮ ᱥᱤᱢᱚ(ᱱ):

ᱚᱯᱥᱚᱨ ᱟ

ᱟ, ᱵ = ᱵ, ᱟ + ᱵ

ᱚᱞ(ᱩᱱᱫᱩᱠ(ᱯᱷᱤᱵᱚᱱᱟᱪᱤ(᱑᱐)))

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱵᱩᱰᱷᱟᱹᱞ\_ᱵᱟᱪᱷᱟᱣ(ᱟᱨᱨᱮ):

ᱱ = ᱡᱮᱞᱮᱝ(ᱟᱨᱨᱮ)

ᱞᱟᱹᱜᱤᱫ ᱤ ᱨᱮ ᱥᱤᱢᱚ(ᱱ):

ᱞᱟᱹᱜᱤᱫ ᱡ ᱨᱮ ᱥᱤᱢᱚ(0, ᱱ-ᱤ-᱑):

ᱡᱩᱫᱤ ᱟᱨᱨᱮ[ᱡ] > ᱟᱨᱨᱮ[ᱡ+᱑]:

ᱟᱨᱨᱮ[ᱡ], ᱟᱨᱨᱮ[ᱡ+᱑] = ᱟᱨᱨᱮ[ᱡ+᱑], ᱟᱨᱨᱮ[ᱡ]

ᱨᱩᱣᱟᱹᱲ ᱟᱨᱨᱮ

ᱚᱞ(ᱵᱩᱰᱷᱟᱹᱞ\_ᱵᱟᱪᱷᱟᱣ([᱖᱔, ᱓᱔, ᱒᱕, ᱑᱒, ᱒᱒, ᱑᱑, ᱙0]))

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱵᱟᱭᱱᱟᱨᱤ\_ᱥᱮᱸᱫᱽᱨᱟ(ᱟᱨᱮ, ᱡᱳᱥ):

ᱞᱟᱛᱟᱨ, ᱪᱮᱴᱟᱱ = ᱐, ᱡᱮᱞᱮᱝ(ᱟᱨᱮ) - ᱑

ᱡᱚᱠᱷᱚᱱ ᱞᱟᱛᱟᱨ <= ᱪᱮᱴᱟᱱ:

ᱛᱟᱞᱟ = (ᱞᱟᱛᱟᱨ + ᱪᱮᱴᱟᱱ) // ᱒

ᱡᱩᱫᱤ ᱟᱨᱮ[ᱛᱟᱞᱟ] == ᱡᱳᱥ:

ᱨᱩᱣᱟᱹᱲ ᱛᱟᱞᱟ

ᱳᱞᱡᱩᱫᱤ ᱟᱨᱮ[ᱛᱟᱞᱟ] < ᱡᱳᱥ:

ᱞᱟᱛᱟᱨ = ᱛᱟᱞᱟ + ᱑

ᱵᱟᱝᱠᱷᱟᱱ:

ᱪᱮᱴᱟᱱ = ᱛᱟᱞᱟ - ᱑

ᱨᱩᱣᱟᱹᱲ -᱑

ᱚᱞ(ᱵᱟᱭᱱᱟᱨᱤ\_ᱥᱮᱸᱫᱽᱨᱟ([᱑, ᱒, ᱓, ᱔, ᱕, ᱖, ᱗], ᱔))

1. **CODE**

**ᱡᱠᱢᱥᱰ = ᱡᱷᱚᱛᱚ ᱠᱷᱚᱱ ᱢᱟᱨᱟᱝ ᱥᱟᱫᱷᱟᱨᱚᱱ ᱰᱤᱣᱟᱭᱤᱡᱟᱨ**

**—---------------------------------------------------------------------------**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱡᱠᱢᱥᱰ(ᱟ, ᱵ):

ᱡᱚᱠᱷᱚᱱ ᱵ:

ᱟ, ᱵ = ᱵ, ᱟ % ᱵ

ᱨᱩᱣᱟᱹᱲ ᱟ

ᱚᱞ(ᱡᱠᱢᱥᱰ(᱔᱘, ᱑᱘))

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱟᱪᱷᱩᱨ\_ᱩᱱᱫᱩᱠ(ᱯᱟᱷᱤᱞ, ᱠ):

ᱠ = ᱠ % ᱡᱮᱞᱮᱝ(ᱯᱟᱷᱤᱞ)

ᱨᱩᱣᱟᱹᱲ ᱯᱟᱷᱤᱞ[-ᱠ:] + ᱯᱟᱷᱤᱞ[:-ᱠ]

ᱚᱞ(ᱟᱪᱷᱩᱨ\_ᱩᱱᱫᱩᱠ([᱑, ᱒, ᱓, ᱔, ᱕], ᱒))

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱟᱫ\_ᱥᱮᱸᱫᱽᱨᱟ(ᱟᱨᱮ):

ᱱ = ᱡᱮᱞᱮᱝ(ᱟᱨᱮ) + ᱑

ᱢᱩᱴ = ᱱ \* (ᱱ + ᱑) // ᱒

ᱨᱩᱣᱟᱹᱲ ᱢᱩᱴ - ᱥᱮᱞᱮᱫ(ᱟᱨᱮ)

ᱚᱞ(ᱟᱫ\_ᱥᱮᱸᱫᱽᱨᱟ([᱑, ᱒, ᱔, ᱕, ᱖]))

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱴᱨᱟᱱᱥᱯᱚᱡᱽ(ᱢᱮᱴᱨᱤᱠᱥ):

ᱨᱩᱣᱟᱹᱲ [[ᱛᱷᱟᱨ[ᱤ] ᱞᱟᱹᱜᱤᱫ ᱛᱷᱟᱨ ᱨᱮ ᱢᱮᱴᱨᱤᱠᱥ] ᱞᱟᱹᱜᱤᱫ ᱤ ᱨᱮ ᱥᱤᱢᱚ(ᱡᱮᱞᱮᱝ(ᱢᱮᱴᱨᱤᱠᱥ[᱐]))]

ᱚᱞ(ᱴᱨᱟᱱᱥᱯᱚᱡᱽ([[᱑, ᱒], [᱓, ᱔], [᱕, ᱖]]))

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱥᱠᱩᱟᱨᱴ(ᱱ):

ᱟᱴᱠᱟᱨ = ᱱ / ᱒

ᱞᱟᱹᱜᱤᱫ \_ ᱨᱮ ᱥᱤᱢᱚ(᱒᱐):

ᱟᱴᱠᱟᱨ = (ᱟᱴᱠᱟᱨ + ᱱ / ᱟᱴᱠᱟᱨ) / ᱒

ᱨᱩᱣᱟᱹᱲ ᱟᱴᱠᱟᱨ

ᱚᱞ(ᱥᱠᱩᱟᱨᱴ(᱑᱖))

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱥᱮᱸᱫᱽᱨᱟ\_ᱪᱩᱨᱟ(ᱟᱨᱮ):

ᱞᱟᱹᱜᱤᱫ ᱤ ᱨᱮ ᱥᱤᱢᱚ(᱑, ᱡᱮᱞᱮᱝ(ᱟᱨᱮ) - ᱑):

ᱡᱩᱫᱤ ᱟᱨᱮ[ᱤ] > ᱟᱨᱮ[ᱤ - ᱑] ᱟᱨ ᱟᱨᱮ[ᱤ] > ᱟᱨᱮ[ᱤ + ᱑]:

ᱨᱩᱣᱟᱹᱲ ᱟᱨᱮ[ᱤ]

ᱨᱩᱣᱟᱹᱲ ᱞᱚᱛᱳᱥᱮ(ᱟᱨᱮ[᱐], ᱟᱨᱮ[-᱑])

ᱚᱞ(ᱥᱮᱸᱫᱽᱨᱟ\_ᱪᱩᱨᱟ([᱑, ᱓, ᱒᱐, ᱔, ᱑, ᱐]))

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱯᱮᱠᱩᱱᱟ\_ᱮᱞ(ᱱ):

ᱨᱩᱣᱟᱹᱲ ᱱ \* (ᱱ + ᱑) // ᱒

ᱚᱞ(ᱯᱮᱠᱩᱱᱟ\_ᱮᱞ(᱕))

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱩᱞᱴᱟᱹᱮᱞ(ᱱ):

ᱩᱞᱴᱟᱹ = ᱐

ᱡᱚᱠᱷᱚᱱ ᱱ > ᱐:

ᱩᱞᱴᱟᱹ = ᱩᱞᱴᱟᱹ \* ᱑᱐ + ᱱ % ᱑᱐

ᱱ //= ᱑᱐

ᱨᱩᱣᱟᱹᱲ ᱩᱞᱴᱟᱹ

ᱚᱞ(ᱩᱞᱴᱟᱹᱮᱞ(᱑᱒᱓᱔᱕))

1. **CODE**

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱞᱮᱠᱷᱟ\_ᱥᱮᱴ\_ᱵᱤᱴᱥ(ᱱ):

ᱞᱮᱠᱷᱟ = 0

ᱡᱚᱠᱷᱚᱱ ᱱ:

ᱞᱮᱠᱷᱟ += ᱱ & ᱑

ᱱ >>= ᱑

ᱨᱩᱣᱟᱹᱲ ᱞᱮᱠᱷᱟ

ᱚᱞ(ᱞᱮᱠᱷᱟ\_ᱥᱮᱴ\_ᱵᱤᱴᱥ(᱑᱙))

1. **CODE**

ᱤ = ᱑

ᱱ = ᱕

ᱡᱚᱠᱷᱚᱱ ᱤ <= ᱑᱐:

ᱚᱞ(ᱥᱴᱨᱤᱝ(ᱱ) + " \* " + ᱥᱴᱨᱤᱝ(ᱤ) + " = " + ᱥᱴᱨᱤᱝ(ᱱ \* ᱤ))

ᱤ += ᱑

1. **CODE**

ᱫᱟᱲᱮ = [᱒\*\*ᱤ ᱞᱟᱹᱜᱤᱫ ᱤ ᱨᱮ ᱥᱤᱢᱚ(᱑᱐)]

ᱚᱞ(ᱫᱟᱲᱮ)

1. **CODE**

ᱜᱚᱱᱚᱝ = ᱠᱚᱢᱥᱮ(᱑᱐, ᱒᱐, ᱕, ᱑᱕)

ᱚᱞ(ᱜᱚᱱᱚᱝ)

1. **CODE**

ᱪᱟᱱᱟᱪ ᱵᱮᱝᱠ\_ᱮᱠᱟᱣᱩᱱᱴ:

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱠᱚᱨᱮᱱ(ᱱᱤᱡᱮᱨ, ᱮᱠᱟᱣᱩᱱᱴ\_ᱫᱷᱟᱨᱤ, ᱵᱮᱞᱮᱱᱥ=᱐):

ᱱᱤᱡᱮᱨ.ᱮᱠᱟᱣᱩᱱᱴ\_ᱫᱷᱟᱨᱤ = ᱮᱠᱟᱣᱩᱱᱴ\_ᱫᱷᱟᱨᱤ

ᱱᱤᱡᱮᱨ.ᱵᱮᱞᱮᱱᱥ = ᱵᱮᱞᱮᱱᱥ

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱡᱚᱢᱟ(ᱱᱤᱡᱮᱨ, ᱴᱟᱠᱟ):

ᱡᱩᱫᱤ ᱴᱟᱠᱟ > ᱐:

ᱱᱤᱡᱮᱨ.ᱵᱮᱞᱮᱱᱥ += ᱴᱟᱠᱟ

ᱚᱞ("ᱡᱚᱢᱟ ᱥᱚᱯᱷᱚᱞ ᱮᱱᱟ. ")

ᱵᱟᱝᱠᱷᱟᱱ:

ᱚᱞ("ᱴᱟᱠᱟ ᱯᱚᱡᱤᱴᱤᱵᱷ ᱦᱩᱭᱩᱜ ᱞᱟᱹᱠᱛᱤᱭᱟ.")

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱩᱱᱫᱩᱠ(ᱱᱤᱡᱮᱨ, ᱴᱟᱠᱟ):

ᱡᱩᱫᱤ ᱴᱟᱠᱟ <= ᱐:

ᱚᱞ("ᱴᱟᱠᱟ ᱯᱚᱡᱤᱴᱤᱵᱷ ᱦᱩᱭᱩᱜ ᱞᱟᱹᱠᱛᱤᱭᱟ.")

ᱳᱞᱡᱩᱫᱤ ᱴᱟᱠᱟ > ᱱᱤᱡᱮᱨ.ᱵᱮᱞᱮᱱᱥ:

ᱚᱞ("ᱵᱟᱝ ᱴᱷᱤᱠ ᱵᱮᱞᱮᱱᱥ.")

ᱵᱟᱝᱠᱷᱟᱱ:

ᱱᱤᱡᱮᱨ.ᱵᱮᱞᱮᱱᱥ -= ᱴᱟᱠᱟ

ᱚᱞ("ᱩᱫᱩᱱᱠ ᱥᱚᱯᱷᱚᱞ ᱮᱱᱟ.")

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱱᱮᱞ\_ᱵᱮᱞᱮᱱᱥ(ᱱᱤᱡᱮᱨ):

ᱚᱞ("ᱱᱤᱛᱚᱜ ᱵᱟᱞᱮᱱᱥ:", ᱱᱤᱡᱮᱨ.ᱵᱮᱞᱮᱱᱥ)

ᱮᱠᱟᱣᱩᱱᱴ = ᱵᱮᱝᱠ\_ᱮᱠᱟᱣᱩᱱᱴ("ᱥᱤᱣᱱᱟᱛᱷ ᱠᱤᱥᱠᱩ", ᱕᱐᱐)

ᱮᱠᱟᱣᱩᱱᱴ.ᱡᱚᱢᱟ(᱒᱐᱐)

ᱮᱠᱟᱣᱩᱱᱴ.ᱩᱱᱫᱩᱠ(᱑᱐᱐)

ᱮᱠᱟᱣᱩᱱᱴ.ᱱᱮᱞ\_ᱵᱮᱞᱮᱱᱥ()

ᱮᱠᱟᱣᱩᱱᱴ.ᱩᱱᱫᱩᱠ(᱗᱐᱐)

ᱮᱠᱟᱣᱩᱱᱴ.ᱱᱮᱞ\_ᱵᱮᱞᱮᱱᱥ()

1. **CODE**

ᱪᱟᱱᱟᱪ ᱟᱛᱢ:

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱠᱚᱨᱮᱱ(ᱱᱤᱡᱮᱨ, ᱯᱤᱱ):

ᱱᱤᱡᱮᱨ.ᱯᱤᱱ = ᱯᱤᱱ

ᱱᱤᱡᱮᱨ.ᱵᱮᱞᱮᱱᱥ = ᱑᱐᱐᱐

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱚᱛᱷᱮᱱᱴᱤᱠᱮᱴ(ᱱᱤᱡᱮᱨ, ᱯᱤᱱ\_ᱳᱞᱢᱮ):

ᱡᱩᱫᱤ ᱯᱤᱱ\_ᱳᱞᱢᱮ == ᱱᱤᱡᱮᱨ.ᱯᱤᱱ:

ᱨᱩᱣᱟᱹᱲ ᱥᱟᱹᱨᱤ

ᱵᱟᱝᱠᱷᱟᱱ:

ᱨᱩᱣᱟᱹᱲ ᱮᱲᱮ

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱱᱮᱞᱢᱮ\_ᱵᱮᱞᱮᱱᱥ(ᱱᱤᱡᱮᱨ):

ᱚᱞ("ᱟᱢᱟᱠ ᱵᱮᱞᱮᱱᱥ:", ᱱᱤᱡᱮᱨ.ᱵᱮᱞᱮᱱᱥ)

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱩᱱᱫᱩᱠ(ᱱᱤᱡᱮᱨ, ᱟᱢᱳᱱᱴ):

ᱡᱩᱫᱤ ᱟᱢᱳᱱᱴ <= ᱐:

ᱚᱞ("ᱜᱚᱱᱚᱝ ᱠᱷᱚᱱ ᱵᱟᱹᱲᱛᱤ ᱦᱩᱭᱩᱜ ᱞᱟᱹᱠᱛᱤᱭᱟ ᱐.")

ᱳᱞᱡᱩᱫᱤ ᱟᱢᱳᱱᱴ > ᱱᱤᱡᱮᱨ.ᱵᱮᱞᱮᱱᱥ:

ᱚᱞ("ᱚᱱᱟ ᱫᱷᱚᱱ ᱫᱳᱷ ᱵᱟᱝ ᱴᱷᱤᱠ ᱫᱷᱚᱱ.")

ᱵᱟᱝᱠᱷᱟᱱ:

ᱱᱤᱡᱮᱨ.ᱵᱮᱞᱮᱱᱥ -= ᱟᱢᱳᱱᱴ

ᱚᱞ("ᱟᱢᱟᱠ ᱛᱟᱠᱟ ᱷᱟᱛᱳᱣ ᱛᱚᱠᱢ, ᱥᱚᱨᱤᱠ ᱟᱠᱟᱱ ᱵᱮᱞᱮᱱᱥ:", ᱱᱤᱡᱮᱨ.ᱵᱮᱞᱮᱱᱥ)

ᱟᱛᱢ = ᱟᱛᱢ(᱑᱒᱓᱔)

ᱯᱤᱱ\_ᱳᱞᱢᱮ = ᱑᱒᱓᱔

ᱡᱩᱫᱤ ᱟᱛᱢ.ᱚᱛᱷᱮᱱᱴᱤᱠᱮᱴ(ᱯᱤᱱ\_ᱳᱞᱢᱮ):

ᱚᱞ("ᱷᱳᱭ ᱮᱱᱟ")

ᱟᱛᱢ.ᱱᱮᱞᱢᱮ\_ᱵᱮᱞᱮᱱᱥ()

ᱟᱛᱢ.ᱩᱱᱫᱩᱠ(᱕᱐᱐)

ᱵᱟᱝᱠᱷᱟᱱ:

ᱚᱞ("ᱵᱟᱝ ᱷᱳᱭ ᱮᱱᱟ")

1. **CODE**

ᱪᱟᱱᱟᱪ ᱵᱳᱛᱥᱤᱥᱛᱢ:

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱠᱚᱨᱮᱱ(ᱱᱤᱡᱮᱨ):

ᱱᱤᱡᱮᱨ.ᱵᱳᱛᱠᱩ = {"ᱥᱮᱣ": ᱐, "ᱠᱟᱭᱨᱟ": ᱐, "ᱱᱮᱢᱩ": ᱐}

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱵᱳᱛ(ᱱᱤᱡᱮᱨ, ᱡᱚ):

ᱡᱩᱫᱤ ᱡᱚ ᱨᱮ ᱱᱤᱡᱮᱨ.ᱵᱳᱛᱠᱩ:

ᱱᱤᱡᱮᱨ.ᱵᱳᱛᱠᱩ[ᱡᱚ] += ᱑

ᱵᱟᱝᱠᱷᱟᱱ:

ᱚᱞ("ᱵᱟᱝ ᱴᱷᱤᱠ ᱵᱟᱪᱷᱟᱣ.")

ᱴᱷᱟᱹᱣᱠᱟᱹ ᱩᱫᱩᱜ\_ᱚᱨᱡᱚ(ᱱᱤᱡᱮᱨ):

ᱚᱞ("ᱵᱳᱛ ᱚᱨᱡᱚ:")

ᱞᱟᱹᱜᱤᱫ ᱡᱚ, ᱵᱳᱛᱠᱩ ᱨᱮ ᱱᱤᱡᱮᱨ.ᱵᱳᱛᱠᱩ.items():

ᱚᱞ(ᱡᱚ, ":", ᱵᱳᱛᱠᱩ)

ᱵᱳᱛ\_ᱥᱤᱥᱛᱢ = ᱵᱳᱛᱥᱤᱥᱛᱢ()

ᱵᱳᱛ\_ᱥᱤᱥᱛᱢ.ᱵᱳᱛ("ᱥᱮᱣ")

ᱵᱳᱛ\_ᱥᱤᱥᱛᱢ.ᱵᱳᱛ("ᱠᱟᱭᱨᱟ")

ᱵᱳᱛ\_ᱥᱤᱥᱛᱢ.ᱵᱳᱛ("ᱥᱮᱣ")

ᱵᱳᱛ\_ᱥᱤᱥᱛᱢ.ᱵᱳᱛ("ᱱᱮᱢᱩ")

ᱵᱳᱛ\_ᱥᱤᱥᱛᱢ.ᱵᱳᱛ("ᱥᱮᱣ")

ᱵᱳᱛ\_ᱥᱤᱥᱛᱢ.ᱩᱫᱩᱜ\_ᱚᱨᱡᱚ()

1. **CODE**

ᱞᱟᱹᱜᱤᱫ ᱤ ᱨᱮ ᱥᱤᱢᱚ(᱑, ᱖):

ᱚᱞ("\*" \* ᱤ)

1. **CODE**

ᱞᱟᱹᱜᱤᱫ ᱤ ᱨᱮ ᱥᱤᱢᱚ(᱕, ᱐, -᱑):

ᱚᱞ("\*" \* ᱤ)

1. **CODE**

ᱞᱟᱹᱜᱤᱫ ᱤ ᱨᱮ ᱥᱤᱢᱚ(᱑, ᱖):

ᱚᱞ(" " \* (᱕ - ᱤ) + "\*" \* (᱒ \* ᱤ - ᱑))

1. **CODE**

ᱱ = ᱕

ᱞᱟᱹᱜᱤᱫ ᱤ ᱨᱮ ᱥᱤᱢᱚ(᱑, ᱱ + ᱑):

ᱚᱞ(" " \* (ᱱ - ᱤ) + "\*" \* (᱒ \* ᱤ - ᱑))

ᱞᱟᱹᱜᱤᱫ ᱤ ᱨᱮ ᱥᱤᱢᱚ(ᱱ - ᱑, ᱐, -᱑):

ᱚᱞ(" " \* (ᱱ - ᱤ) + "\*" \* (᱒ \* ᱤ - ᱑))

**THE END**