Python Basics

Highlights

Example

Question

Notes

Coding Style

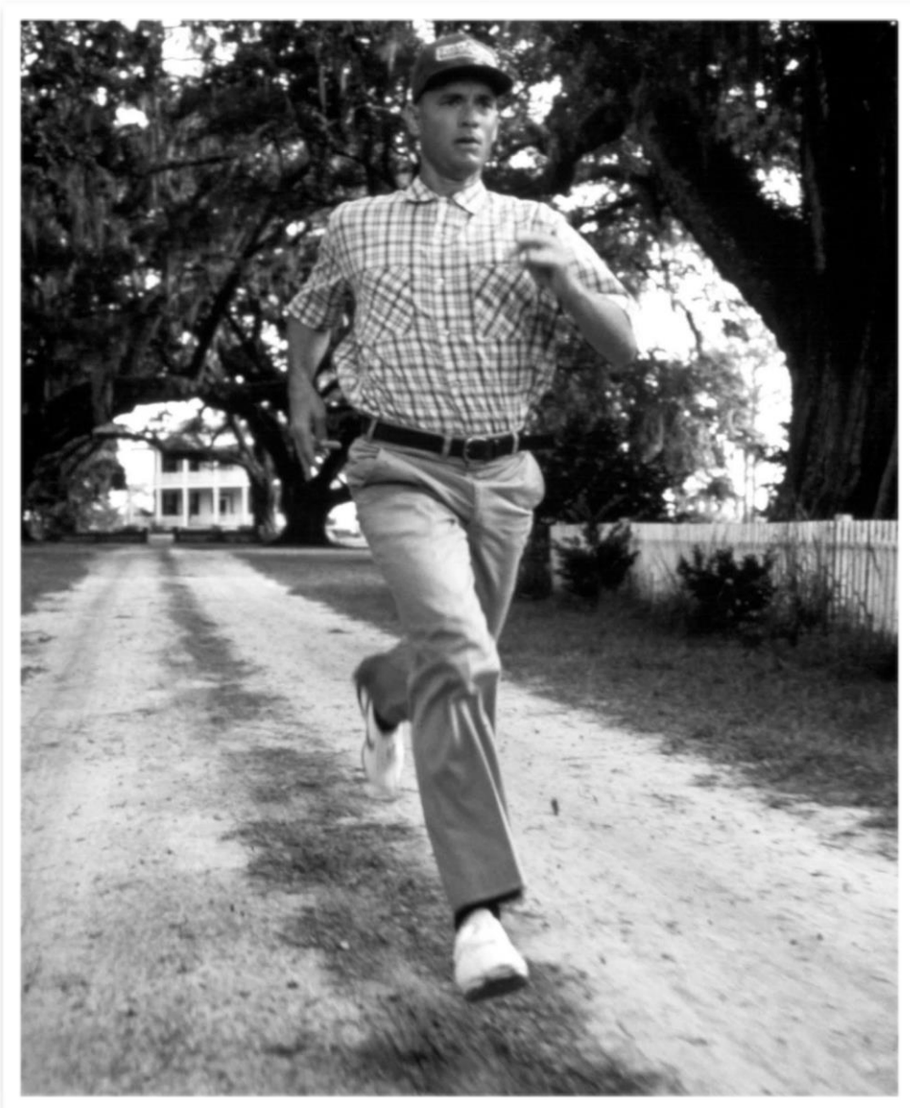
Roadmap

1. **Introduction**
2. Objects, Variables, and Data Types
3. Strings, List and Dictionary
4. Control Flow Statements

Run! Python, Run!

I just feel like running……

Python!



Run! Python, Run!

- Installation
  - Python
  - **Jupyter Notebook (Anaconda)**
  - IDEs (Pycharm/Spyder)

Run! Python, Run!

- Interactive Shell Combining Code with Text
  - ▶ Try Python as a scientific calculator
    - ✓ Python arithmetic operators
    - ✓ Python code is executed line by line (cell by cell)

```
In [1]: 1 + 2
Out[1]: 3

In [2]: (2+3.5) * 6 / (1.75-0.25)
Out[2]: 22.0

In [3]: 1.5 ** 3   # ** 1.5 to the power of 3
Out[3]: 3.375
```

Run! Python, Run!

- Interactive Shell Combining Code with Text
  - ▶ Try Python as a scientific calculator
    - ✓ Python arithmetic operators
    - ✓ Python code is executed line by line (cell by cell)

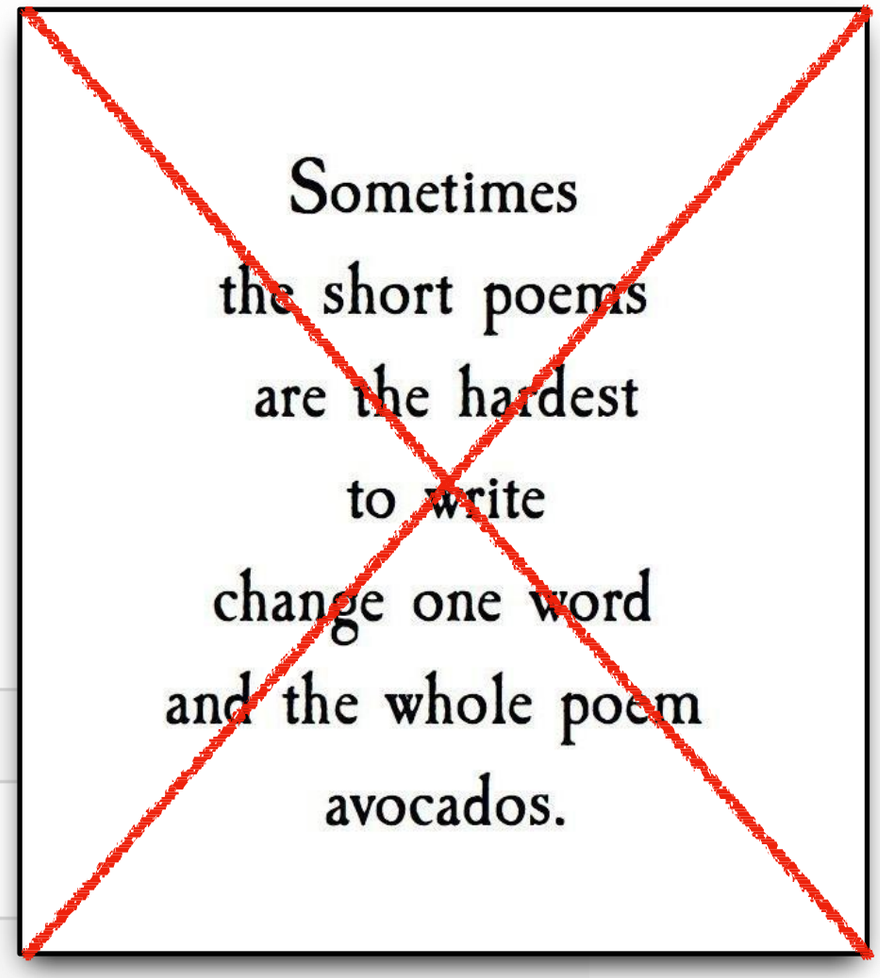
```
In [1]: 1 + 2
Out[1]: 3

In [2]: (2+3.5) * 6 / (1.75-0.25)
Out[2]: 22.0

In [3]: 1.5 ** 3   # ** 1.5 to the power of
Out[3]: 3.375
```

Notes

Splitting a statement into multiple lines is not allowed.

Run! Python, Run!

- Interactive Shell Combining Code w
  - ▸ Try Python as a scientific calculator
    - ✓ Python arithmetic operators
    - ✓ Python code is executed line by line (c




```
In [1]: 1 + 2
Out[1]: 3

In [2]: (2+3.5) * 6 / (1.75-0.25)
Out[2]: 22.0

In [3]: 1.5 ** 3   # ** 1.5 to the power of
Out[3]: 3.375
```

Not allowed!!

Notes

Splitting a statement into multiple lines is not allowed.

Run! Python, Run!

- Interactive Shell Combining Code with Text
  - ▶ Try Python as a scientific calculator
    - ✓ Python arithmetic operators
    - ✓ Python code is executed line by line (cell by cell)

Example 1

A grocery store had 500 packs of cookies in storage. Among the total storage, 5% were dumped due to expiration, and another 230 packs were sold. Calculate how many packs of cookies are left in the grocery store.

```
In [4]: 500*(1-0.05) - 230
```

```
Out[4]: 245.0
```

Roadmap

1. Introduction
2. **Objects, Variables, and Data Types**
3. Strings, List and Dictionary
4. Control Flow Statements

Objects, Variables, and Data Types

Objects, Variables, and Data Types

An Overview of Datatypes:

Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	`str`
Numeric Types:	`int`, `float`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Boolean Type:	`bool`

https://www.w3schools.com/python/python_datatypes.asp

Objects, Variables, and Data Types

- Objects
  - Almost everything in Python is an object
  - Every object has a value and a type

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - ▸ Variables gives names to objects
    - ✓ Reuse the same value
    - ✓ Improve the readability
    - ✓ Enable control flow

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Assignment operator “="

Example 1

A grocery store had 500 packs of cookies in storage. Among the total storage, 5% were dumped due to expiration, and another 230 packs were sold. Calculate how many packs of cookies are left in the grocery store.

```
In [5]: storage = 500
        dump_rate = 0.05
        sold = 230

        storage*(1-dump_rate) - sold
```

```
Out[5]: 245.0
```

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Variable(s): name(s) on the left

Example 1

A grocery store had 500 packs of cookies in storage. Among the total storage, 5% were dumped due to expiration, and another 230 packs were sold. Calculate how many packs of cookies are left in the grocery store.

In [5]:
```
[redacted] = 500
[redacted] = 0.05
[redacted] = 230

storage*(1-dump_rate) - sold
```

Out[5]: 245.0

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Variable(s): name(s) on the left

Example 1

A grocery store had 500 packs of cookies in storage. Among the total storage, 5% were dumped due to expiration, and another 230 packs were sold. Calculate how many packs of cookies are left in the grocery store.

```
In [5]: [redacted] = 500
        [redacted] = 0.05
        [redacted] = 230

        storage*(1-dump_rate) - sold

Out[5]: 245.0
```

Notes

- **Only one “word”**
- **Only consist of letters, numbers, and underscores**
- **Cannot begin with a number**
- **Avoid contradictions with Python keywords**

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Value(s): expression(s) on the right

Example 1

A grocery store had 500 packs of cookies in storage. Among the total storage, 5% were dumped due to expiration, and another 230 packs were sold. Calculate how many packs of cookies are left in the grocery store.

In [5]:
```
storage = 500
dump_rate = 0.05
sold = 230

storage*(1-dump_rate) - sold
```

Out[5]:
```
245.0
```

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - The value of a variable can be retrieved by invoking the name

Example 1

A grocery store had 500 packs of cookies in storage. Among the total storage, 5% were dumped due to expiration, and another 230 packs were sold. Calculate how many packs of cookies are left in the grocery store.

In [5]:
```
storage = 500
dump_rate = 0.05
sold = 230

```

Out[5]: 245.0

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - The value of a variable can be retrieved by invoking the name

Example 1

A grocery store had 500 packs of cookies in storage. Among the total storage, 5% were dumped due to expiration, and another 230 packs were sold. Calculate how many packs of cookies are left in the grocery store.

```
In [5]: storage = 500
        dump_rate = 0.05
        sold = 230

        storage*(1-dump_rate) - sold
```

```
Out[5]: 245.0
```

Coding Style

PEP 8 Style Guide:

- **Using empty lines for better readability**

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Assign new values to variables

```
In [5]: storage = 500
        dump_rate = 0.05
        sold = 230

        storage*(1-dump_rate) - sold
```

```
Out[5]: 245.0
```

```
In [6]: """Storage is updated with a new value"""
```

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Assign new values to variables

In [5]:
```
storage = 500
dump_rate = 0.05
sold = 230

storage*(1-dump_rate) - sold
```

Out[5]: 245.0

In [6]:
```
"""Storage is updated with a new value"""

storage = storage*(1-dump_rate) - sold
```

500

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Assign new values to variables

```
In [5]: storage = 500
        dump_rate = 0.05
        sold = 230

        storage*(1-dump_rate) - sold
```

```
Out[5]: 245.0
```

```
In [6]: """Storage is updated with a new value"""

        [          ] = storage*(1-dump_rate) - sold
```

The new value 245 is assigned as the old one is forgotten

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Assign new values to variables

```
In [5]: storage = 500
        dump_rate = 0.05
        sold = 230

        storage*(1-dump_rate) - sold
```

```
Out[5]: 245.0
```

```
In [6]: """Storage is updated with a new value"""

        storage = storage*(1-dump_rate) - sold
```

Coding Style

PEP 8 Style Guide:

- **Function and variable names**
- **Whitespace in expressions and statements**

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Assign values of function outputs to variables

Example 2

Use the “input” function in assignment statements

In [13]:
```
greetings = 'Hello '
name = input('Key in your name: ')  # Assign the output of input to name

print(greetings + name)             # Print both greetings and name
```

```
Key in your name: Peter
Hello Peter
```

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Assign values of function outputs to variables

Example 2

Use the “input” function in assignment statements

In [13]:
```
greetings = 'Hello '
name = input('Key in your name: ')  # Assign the output of input to name

print(greetings + name)             # Print both greetings and name
```

```
Key in your name: Peter
Hello Peter
```

Functions: a block of organized, reusable code that is used to perform a single, related action

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Assign values of function outputs to variables

Example 2

Use the “input” function in assignment statements

In [13]:
```
greetings = 'Hello '
                                        # Assign the output of input to name

print(greetings + name)                 # Print both greetings and name
```

```
Key in your name: Peter
Hello Peter
```

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Assign values of function outputs to variables

Example 2

Use the “input” function in assignment statements

```
In [13]: greetings = 'Hello '
         name =                                  # Assign the output of input to name

         print(greetings + name)                 # Print both greetings and name
```

```
Key in your name: Peter
Hello Peter
```

Output of the input function is the text given by users

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Assign values of function outputs to variables

Example 2

Use the "input" function in assignment statements

```
In [13]: greetings = 'Hello '
         name = input(                    )   # Assign the output of input to name

         print(greetings + name)              # Print both greetings and name
```

```
                          Peter
Hello Peter
```

Message displayed when users key in the text

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Assign values of function outputs to variables

Example 2

Use the “input” function in assignment statements

In [13]:
```
greetings = 'Hello '
name = input('Key in your name: ')  # Assign the output of input to name

print(          +      )            # Print both greetings and name
```

```
Key in your name: Peter
```

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Python vs Math

Question

Which assignment statement is correct?

A. x + y = 2

B. x * y = 1

C. 2 = x

D. xy = 2

E. None of the above is correct

Objects, Variables, and Data Types

- Variables and Assignment Statements
  - Other assignment operations

Operators	Remarks	Examples
[redacted]	Equivalent to `x = x + y`	`x += y`
[redacted]	Equivalent to `x = x - y`	`x -= y`
[redacted]	Equivalent to `x = x * y`	`x *= y`
[redacted]	Equivalent to `x = x / y`	`x /= y`
[redacted]	Equivalent to `x = x**y`	`x **= y`

Objects, Variables, and Data Types

- Basic Built-in Data Types
  - Various types of data may be different in nature
  - Data types determine how functions and operators work
  - Data types can be retrieved by function [redacted]

Objects, Variables, and Data Types

- Basic Built-in Data Types
  - Numeric types
    - ✓ [redacted]: signed integers
    - ✓ [redacted]: floating point real numbers
    - ✓ [redacted] and [redacted] long integers and complex numbers (not covered in this course)

Objects, Variables, and Data Types

- Basic Built-in Data Types
  - Numeric types

```
In [10]: storage = 5000
         type(storage)
```

```
Out[10]: int
```

```
In [11]: storage = 5000.0
         type(storage)
```

```
Out[11]: float
```

```
In [12]: storage = 5000
         dump_rate = 0.05

         type(storage * (1 - dump_rate))
```

```
Out[12]: float
```

Objects, Variables, and Data Types

- Basic Built-in Data Types
  - String: [redacted]

    ✓ Single or double quotation marks used in defining strings

    ✓ Operators [redacted] and [redacted] can be used to manipulate strings

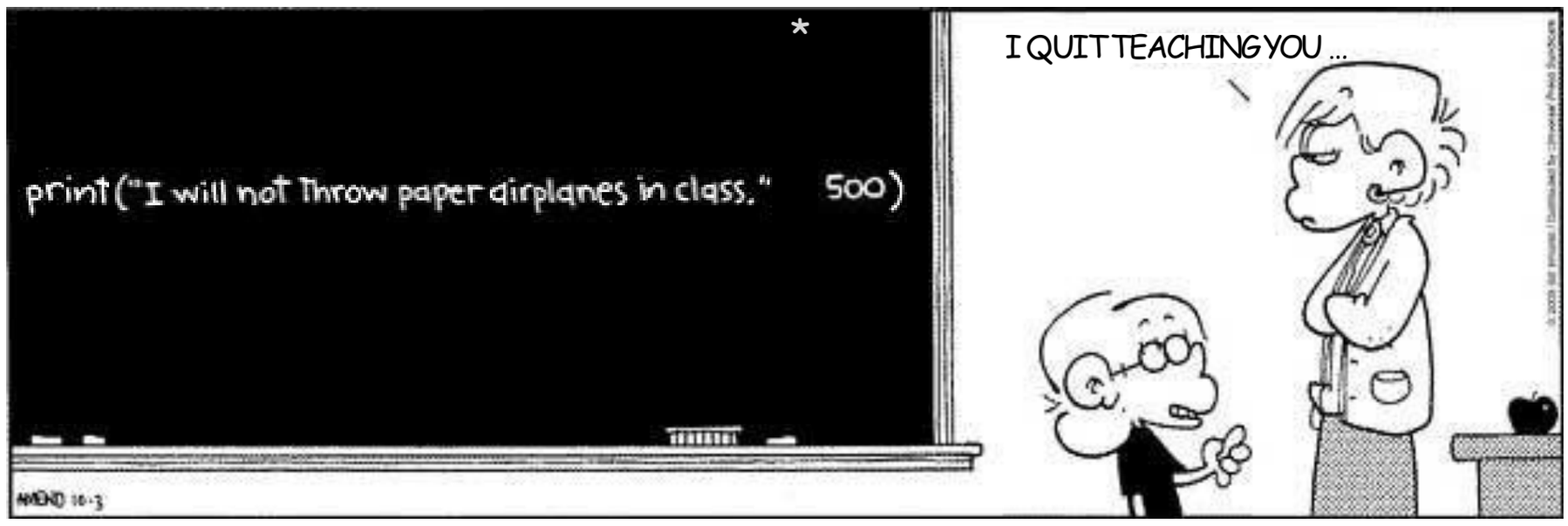
```
In [10]: greeting = 'Hello DAO2702'

         type(greeting)

Out[10]: str

In [11]: storage_str = "5000.0"

         type(storage_str)

Out[11]: str
```

Objects, Variables, and Data Types

- Basic Built-in Data Types
  - String: [redacted]

✓ Single or double quotation marks used in defining strings

✓ Operators [redacted] and [redacted] can be used to manipulate strings



Objects, Variables, and Data Types

- Basic Built-in Data Types
  - String: [redacted]

Example 3

The function "input" enables users to type in a string of characters. Print a message "You input is:", followed by the input string repeated ten times.

```
In [15]: input_str = input('Type something: ')

         print('Your input is: ' + input_str*10)

         Type something: 3
```

Objects, Variables, and Data Types

- Basic Built-in Data Types
  - String: [redacted]

Example 3

The function "input" enables users to type in a string of characters. Print a message "You input is:", followed by the input string repeated ten times.

```
In [15]: input_str = input('Type something: ')

         print('Your input is: ' [redacted])
```

```
Type something: 3
Your input is: 3333333333
```

input_str is a string, so "+" combines strings, and "*" replicates strings.

Objects, Variables, and Data Types

- Basic Built-in Data Types
  - String: [redacted]

HOW ABOUT MULTIPLYING THE INPUT NUMBER BY TEN?

```
In [15]: input_str = input('Type something: ')

         print('Your input is: ' + input_str*10)
```

```
Type something: 3
Your input is: 3333333333
```

Objects, Variables, and Data Types

- Data Type Conversion
  - Enable manipulating data differently
  - Use type name as the conversion function

DATA TYPE CONVERSION!!

Objects, Variables, and Data Types

- Data Type Conversion
  - Enable manipulating data differently
  - Use type name as the conversion function

DATA TYPE CONVERSION!!

```
In [16]: input_str = input('Type something: ')

         print('Ten times of your input is: ' + str(float(input_str)*10))
```

```
Type something: 3
Ten times of your input is: 30.0
```

Objects, Variables, and Data Types

- Data Type Conversion
  - Enable manipulating data differently
  - Use type name as the conversion function

DATA TYPE CONVERSION!!

```
In [16]: input_str = input('Type something: ')

         print('Ten times of your input is: ' + str(input_str)*10))
```

```
Type something: 3
Ten times of your input is: 30.0
```

Convert the string to a floating point number

Objects, Variables, and Data Types

- Data Type Conversion
  - Enable manipulating data differently
  - Use type name as the conversion function

DATA TYPE CONVERSION!!

```
In [16]: input_str = input('Type something: ')

         print('Ten times of your input is: ' + [redacted])
```

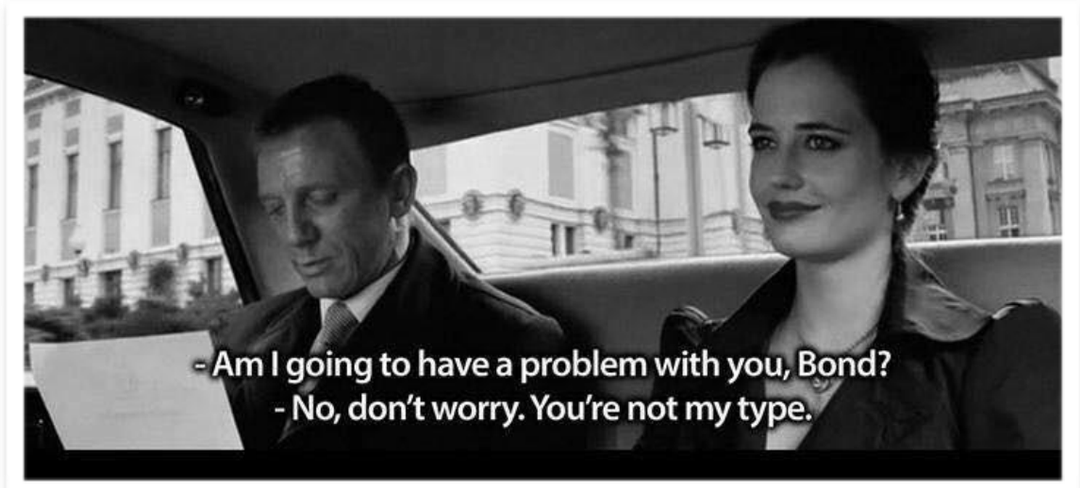
```
Type something: 3
Ten times of your input is: 30.0
```

Convert the floating point number back to a string

Objects, Variables, and Data Types

- Data Type Conversion

BEFORE TYPE CONVERSION



Objects, Variables, and Data Types

- Data Type Conversion

BEFORE TYPE CONVERSION

AFTER TYPE CONVERSION

Objects, Variables, and Data Types

- Data Type Conversion

Question

What is the output message of the following code?

```
a = 3.5
b = 1

print(str(a)*2 + str(b))
```

A. 8

B. 3.53.51

C. 71

D. 3.53.51.0

Roadmap

1. Introduction
2. Objects, Variables, and Data Types
3. **Strings, List and Dictionary**
4. Control Flow Statements

Strings

—“Three glasses” in *Inglourious Basterds*

Strings

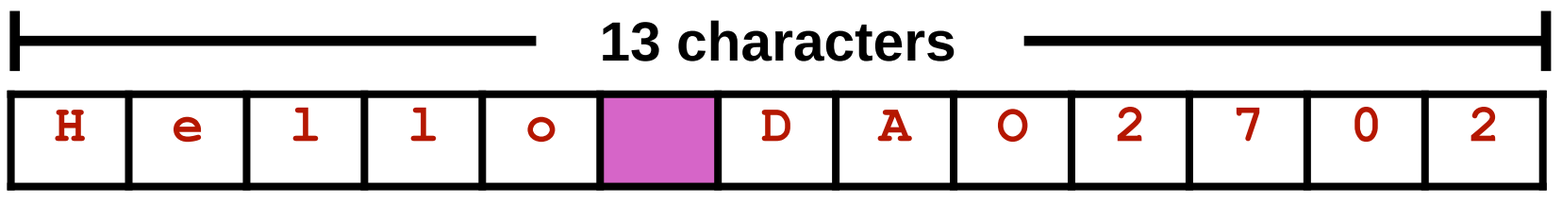
—“Three glasses” for Python programmers

Strings

- Strings are collections of characters
  - Length of a string (the number of characters)

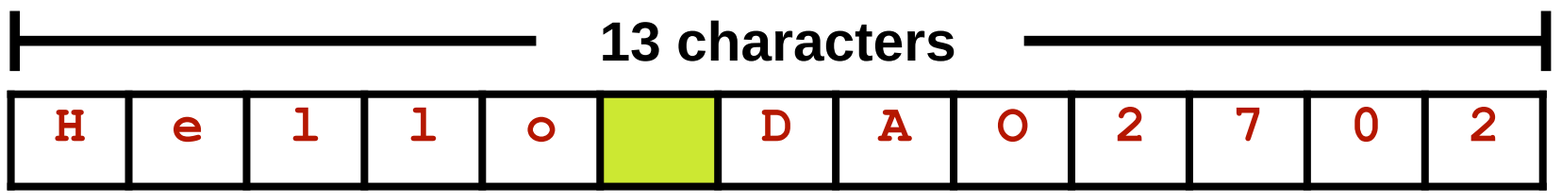
```
In [3]: """Length of the string"""
        len("Hello DAO2702")
```

```
Out[3]: 13
```



Strings

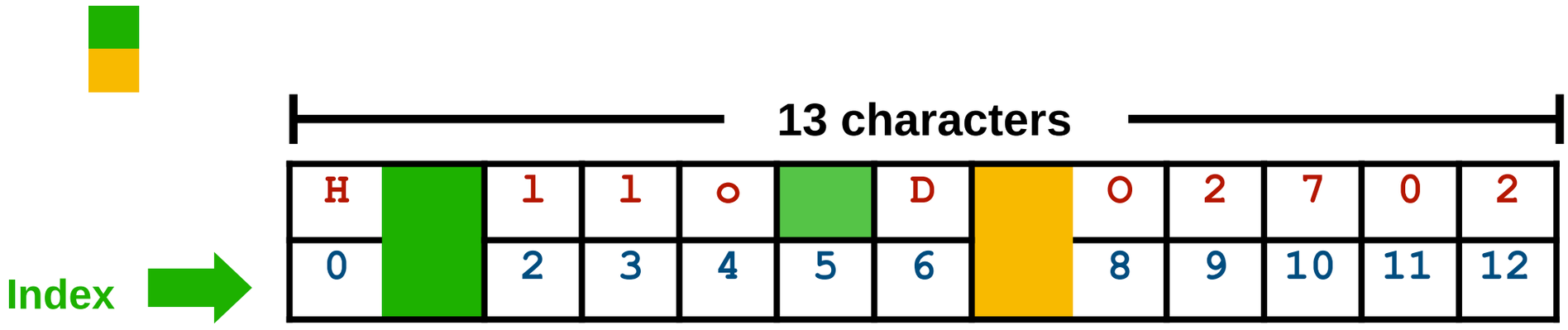
- Strings are collections of characters
  - Accessing individual characters



Strings

- Strings are collections of characters
  - Accessing individual characters

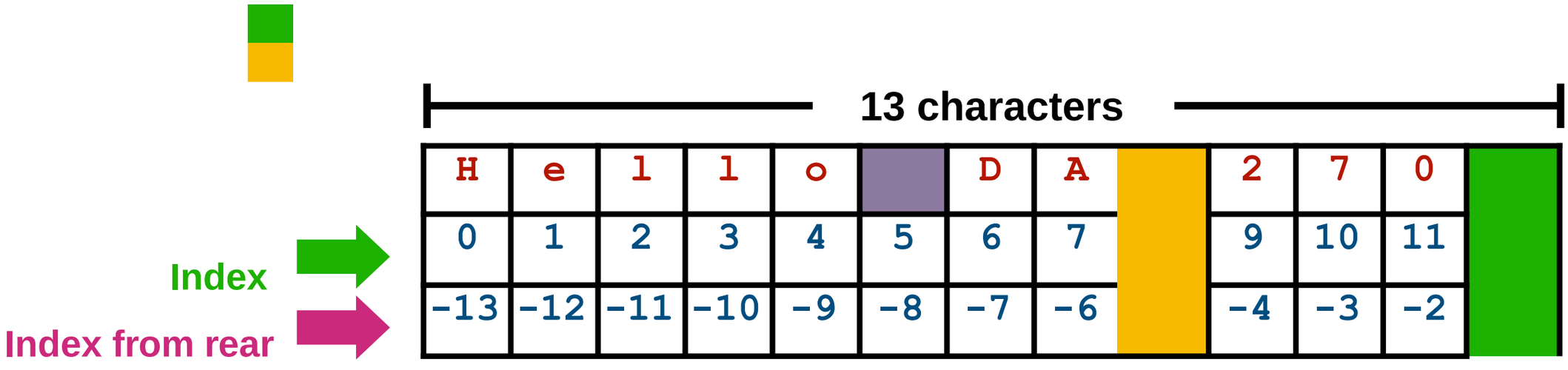
```
In [5]: greetings = "Hello DAO2702"

        letter_e = greetings[1]       # Access the 2nd character "e"
        print(letter_e)

        letter_A = greetings[7]       # Access the 8th character "A"
        print(letter_A)
```



Strings

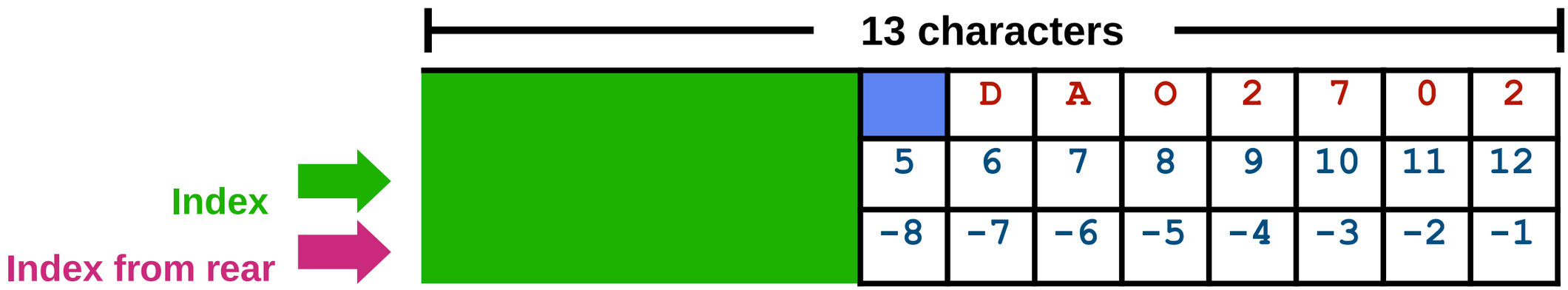
- Strings are collections of characters
  - Accessing individual characters

```
In [6]: greetings = "Hello DAO2702"

letter_2 = greetings[-1]       # Access the last character "2"
print(letter_2)

letter_0 = greetings[-5]       # Access the character "0"
print(letter_0)
```



Strings

- Strings are collections of characters
  - Slicing expression: access a subset of a string



Strings

- Strings are collections of characters
  - Slicing expression: access a subset of a string

`[`***`start`***`:`***`stop`***`:`***`step`***`]`

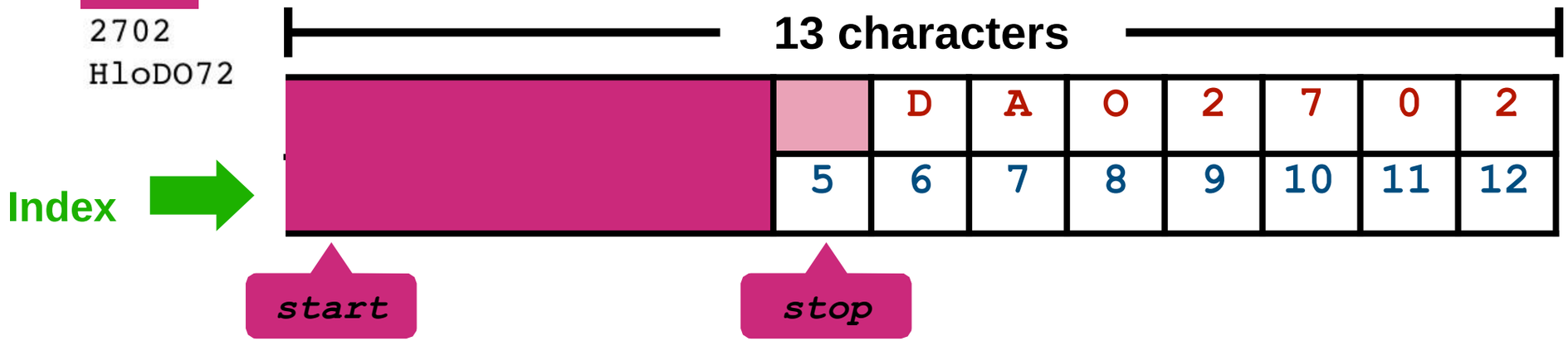
Arguments	Remarks	Default Values
`start`	The first index of the slice	0
`stop`	The index before which the slice stops	Length of the string
`step`	The step length of the slice	1

Strings

- Strings are collections of characters
  - Slicing expression: access a subset of a string

```
In [10]: greetings = "Hello DAO2702"

         [redacted]      # Print the first five characters

         print(greetings[9:13:1])    # Print the last four characters

         print(greetings[0:13:2])    # Print the 1st, 3rd, 5th, ... characters
```

[redacted]
2702
HloDO72

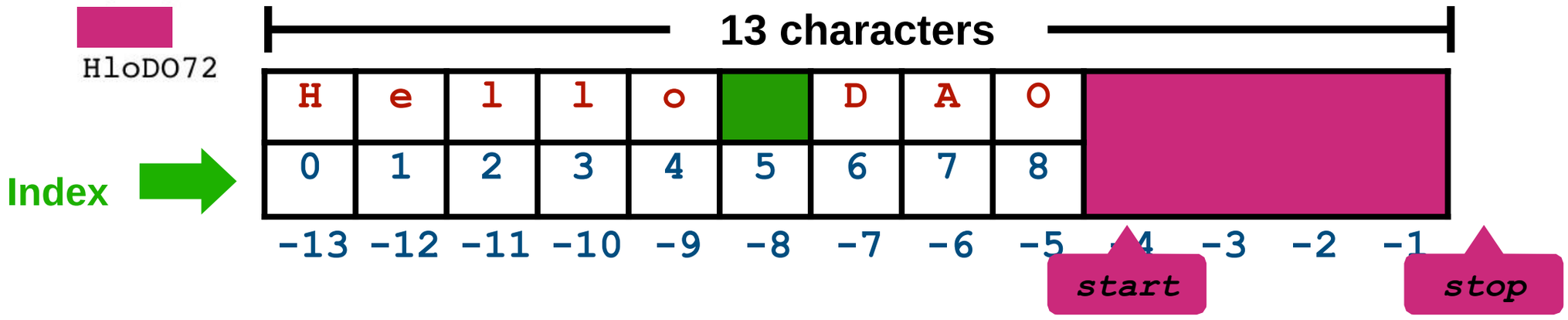


Strings

- Strings are collections of characters
  - Slicing expression: access a subset of a string

```
In [10]: greetings = "Hello DAO2702"

         print(greetings[0:5:1])      # Print the first five characters

         [redacted]                   # Print the last four characters

         print(greetings[0:13:2])     # Print the 1st, 3rd, 5th, ... characters
```

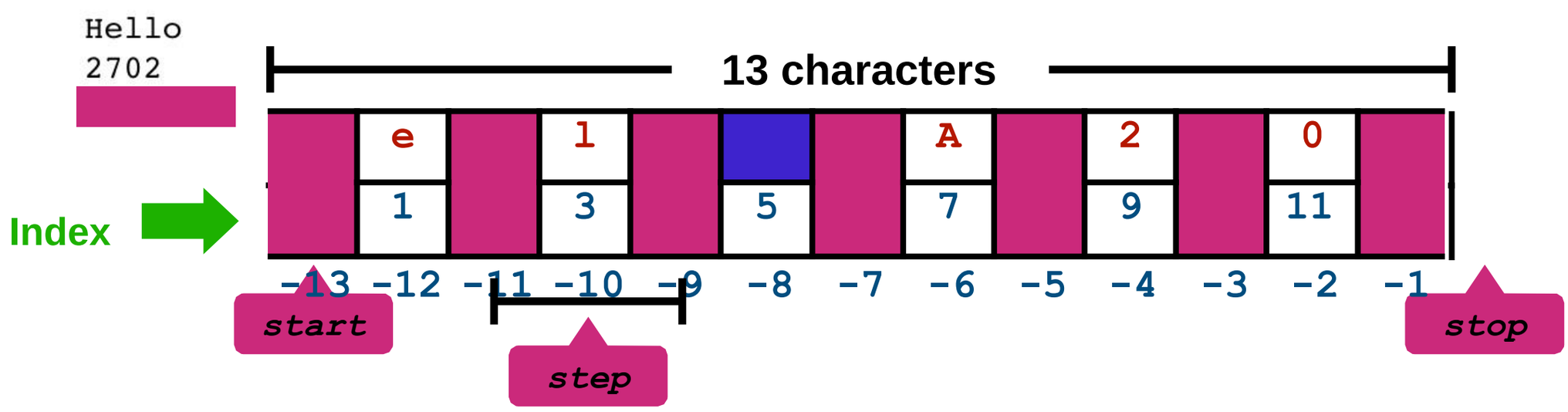
```
Hello
[redacted]
HloDO72
```



Strings

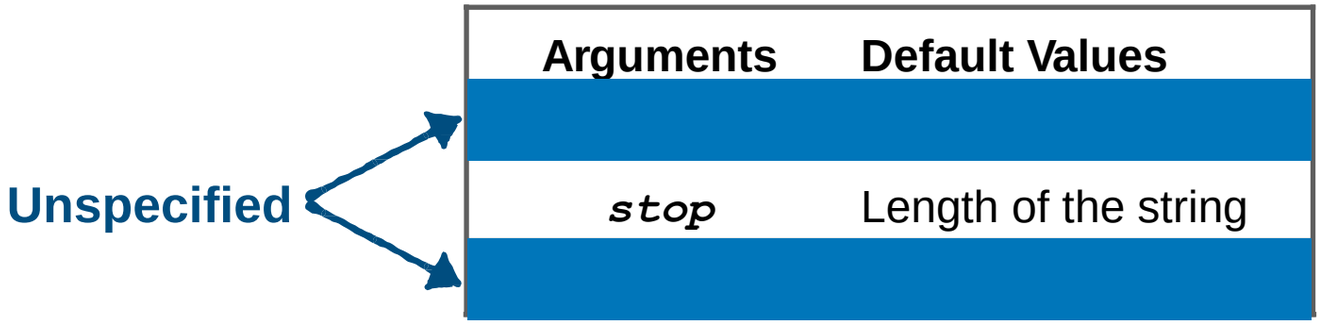
- Strings are collections of characters
  - Slicing expression: access a subset of a string

```
In [10]: greetings = "Hello DAO2702"

         print(greetings[0:5:1])      # Print the first five characters

         print(greetings[9:13:1])    # Print the last four characters

         [redacted]                   # Print the 1st, 3rd, 5th, ... characters
```

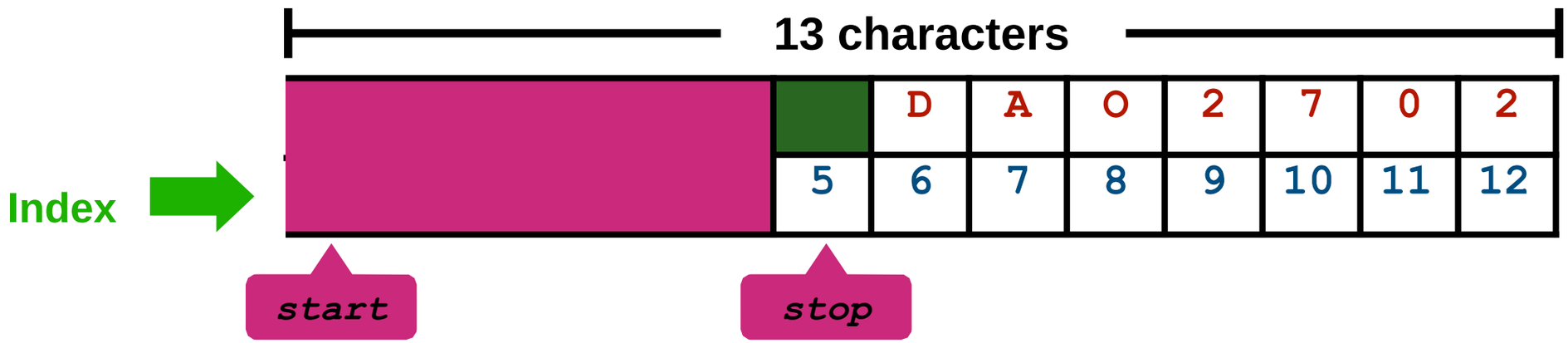


Strings

- Strings are collections of characters
  - Slicing expression

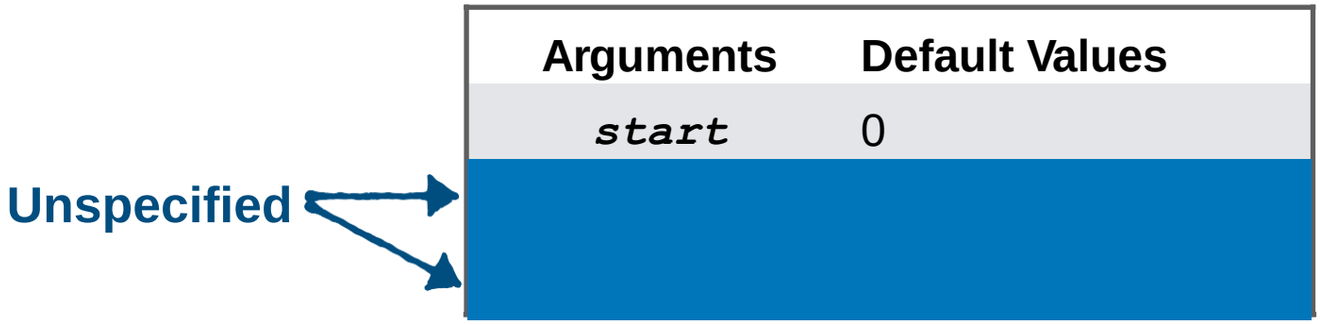


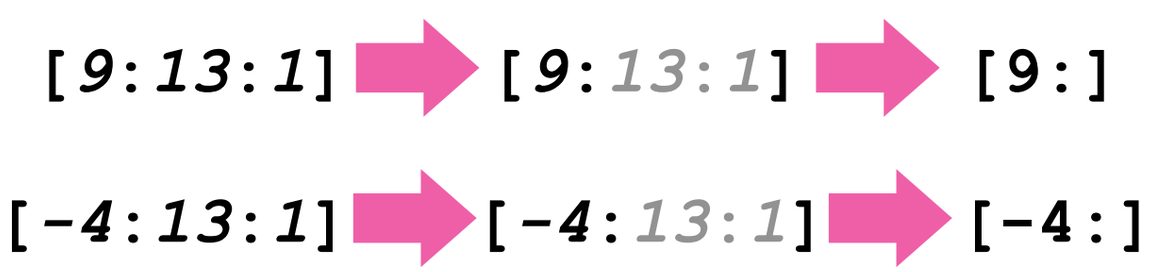
`[0:5:1]` → `[0:5:1]` → `[:5]`

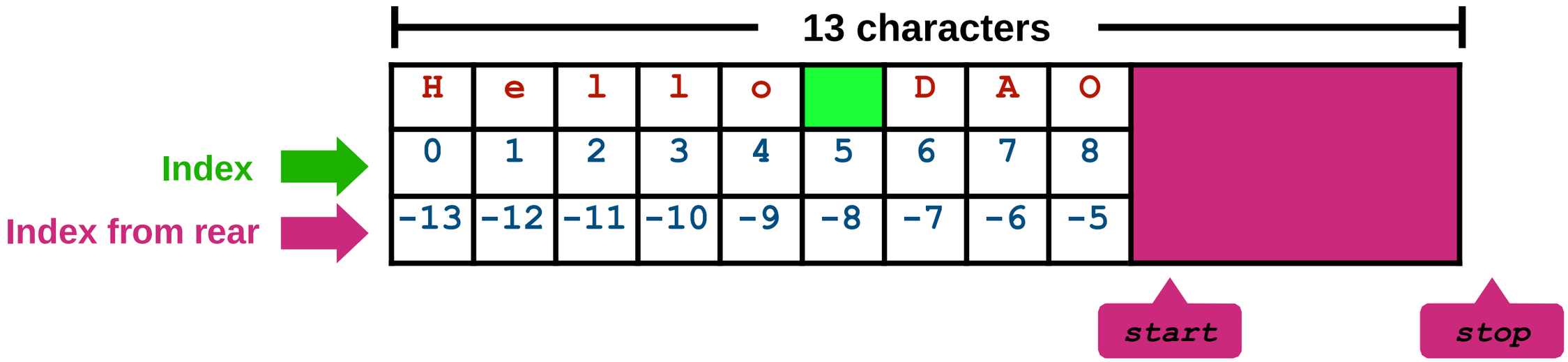


Strings

- Strings are collections of characters
  - Slicing expression

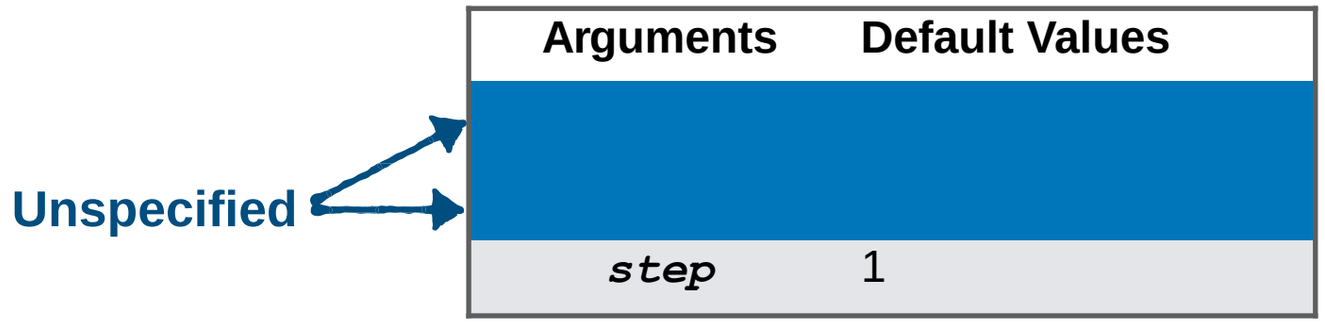




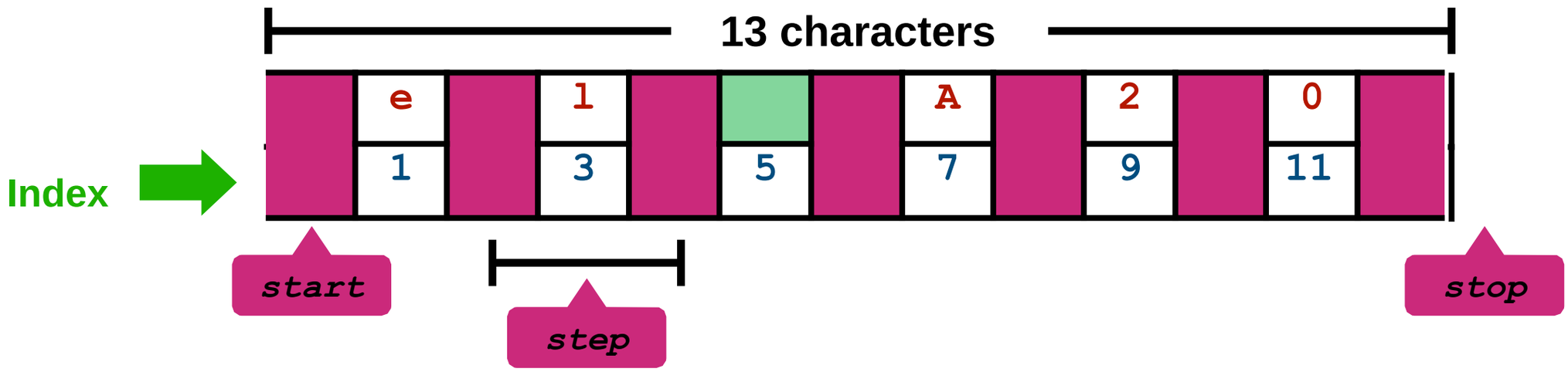


Strings

- Strings are collections of characters
  - Slicing expression



`[0:13:2]` → `[0:13:2]` → `[::2]`



Strings

- Strings are collections of characters
  - Slicing expression: access a subset of a string

Notes

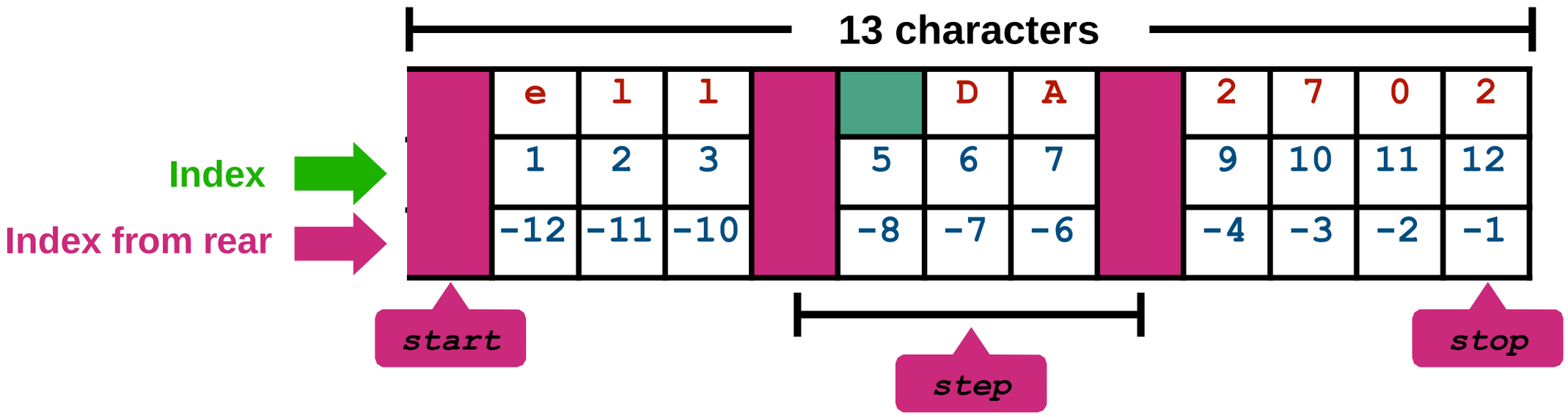
A common mistake is assuming that the last index of the slice is `stop`. Please be very careful that the last index should be the last available integer before `stop`.

Strings

- Strings are collections of characters
  - Slicing expression: access a subset of a string

```
In [8]: greetings = "Hello DAO2702"

        print(greetings[:-1:4])    # Print the 1st, the 5th, and the 9th characters
```

```
HoO
```



Strings

- Strings are collections of characters
  - Indexing and slicing expression

Question

What is the printed message of the following code?

```
sentence = "an easy quiz"

print(sentence[-8:-2])
```

A. easy q

B. easy

C. asy qu

D. asy qui

Strings

- Strings are collections of characters
  - Iterate characters via [redacted]
    - ✓ [redacted] is a built-in data type that defines a sequence of integers
    - ✓ It is used in the form of [redacted]

Arguments	Remarks	Default Values
start	The first index of the integer sequence	0
stop	The index before which the sequence stops	-
step	The step length of the integer sequence	1

Strings

- Strings are collections of characters
  - Iterate characters via [redacted]
    - ✓ [redacted] is a built-in data type that defines a sequence of integers
    - ✓ It is used in the form of [redacted]
    - ✓ Examples

```
range(0, 5, 1)  →  0, 1, 2, 3, 4
range(0, 3)     →  0, 1, 2
range(6)        →  0, 1, 2, 3, 4, 5
```

Strings

- Strings are collections of characters
  - ▶ Iterate characters via [redacted]
    - ✓ Use [redacted] as the index of characters

```
In [14]: name = input("What is your name? ")

for index in range(len(name)):  # Iterate integers 0, 1, ..., len(name)-1
    letter = name[index]
    if letter == ' ':
        continue

    print("Give me a " + letter + ": ")
    print(letter + "!!!")

print("What's that spell?")
print(name + "!!")

print("Go! Go! " + name + "!!")
```

Strings

- Strings are collections of characters
  - ▶ Iterate characters via [redacted]
    - ✓ Use [redacted] as the index of characters

```
In [14]: name = input("What is your name? ")

         for [redacted]:  # Iterate integers 0, 1, ..., len(name)-1
             letter = name[index]
             if letter == ' ':
                 continue

             print("Give me a " + letter + ": ")
             print(letter + "!!!")

         print("What's that spell?")
         print(name + "!!")

         print("Go! Go! " + name + "!!")
```

name

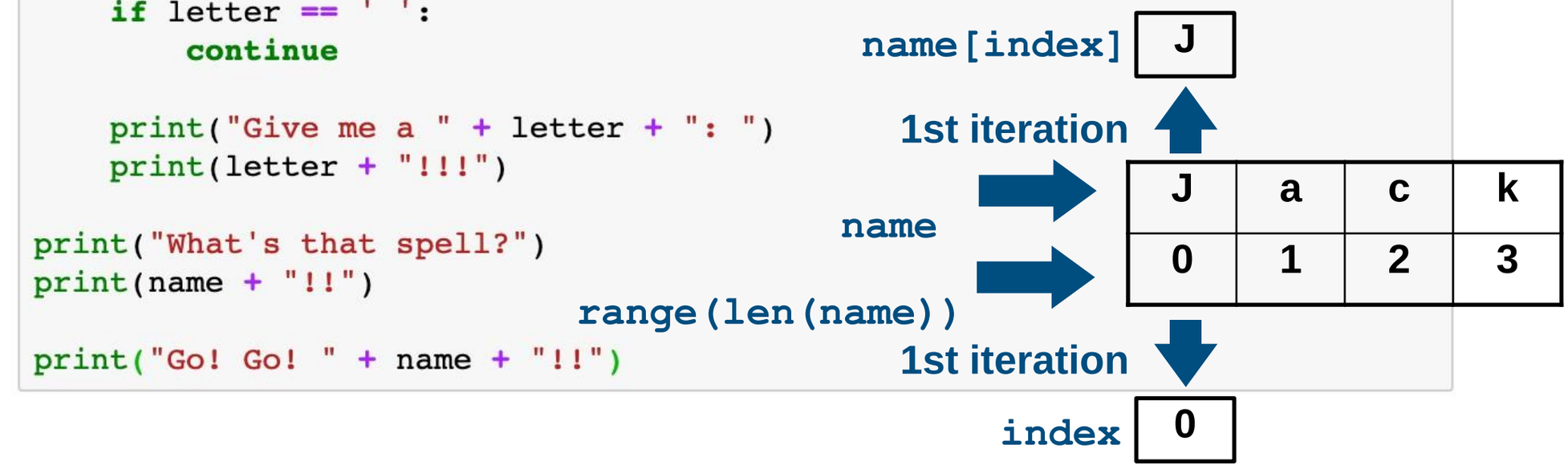
range(len(name))

J	a	c	k
0	1	2	3

Strings

- Strings are collections of characters
  - ▶ Iterate characters via [redacted]
    - ✓ Use [redacted] as the index of characters

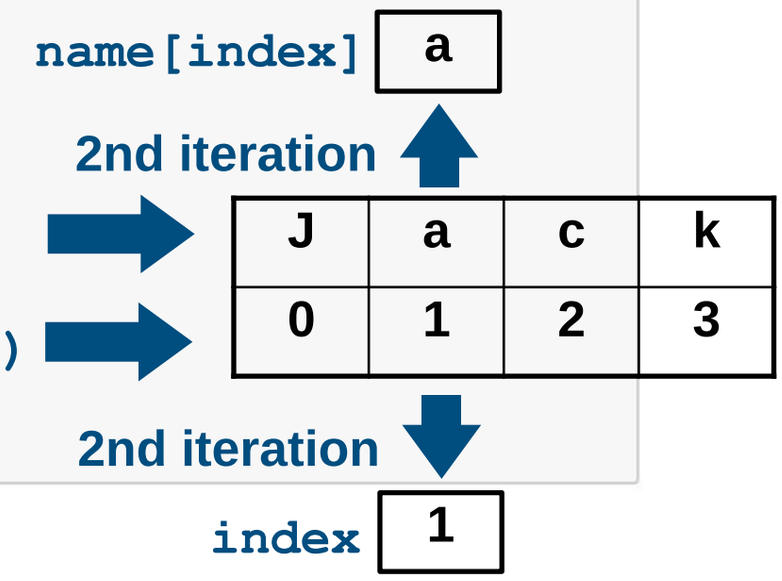
```
In [14]: name = input("What is your name? ")

         for index in range(len(name)):  # Iterate integers 0, 1, ..., len(name)-1
             [redacted]
             if letter == ' ':
                 continue

             print("Give me a " + letter + ": ")
             print(letter + "!!!")

         print("What's that spell?")
         print(name + "!!")

         print("Go! Go! " + name + "!!")
```



Strings

- Strings are collections of characters
  - ▶ Iterate characters via [redacted]
    - ✓ Use [redacted] as the index of characters

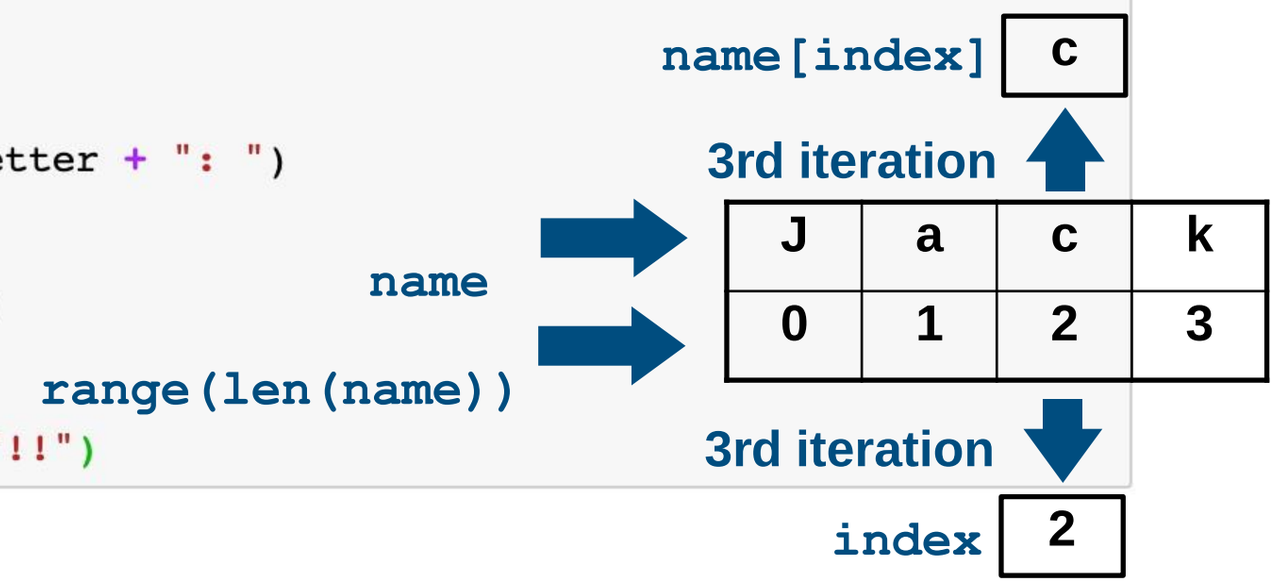
```
In [14]: name = input("What is your name? ")

for index in range(len(name)):  # Iterate integers 0, 1, ..., len(name)-1
    [redacted]
    if letter == ' ':
        continue

    print("Give me a " + letter + ": ")
    print(letter + "!!!")

print("What's that spell?")
print(name + "!!")

print("Go! Go! " + name + "!!")
```



Strings

- Strings are collections of characters
  - ▶ Iterate characters via [redacted]
    - ✓ Use [redacted] as the index of characters

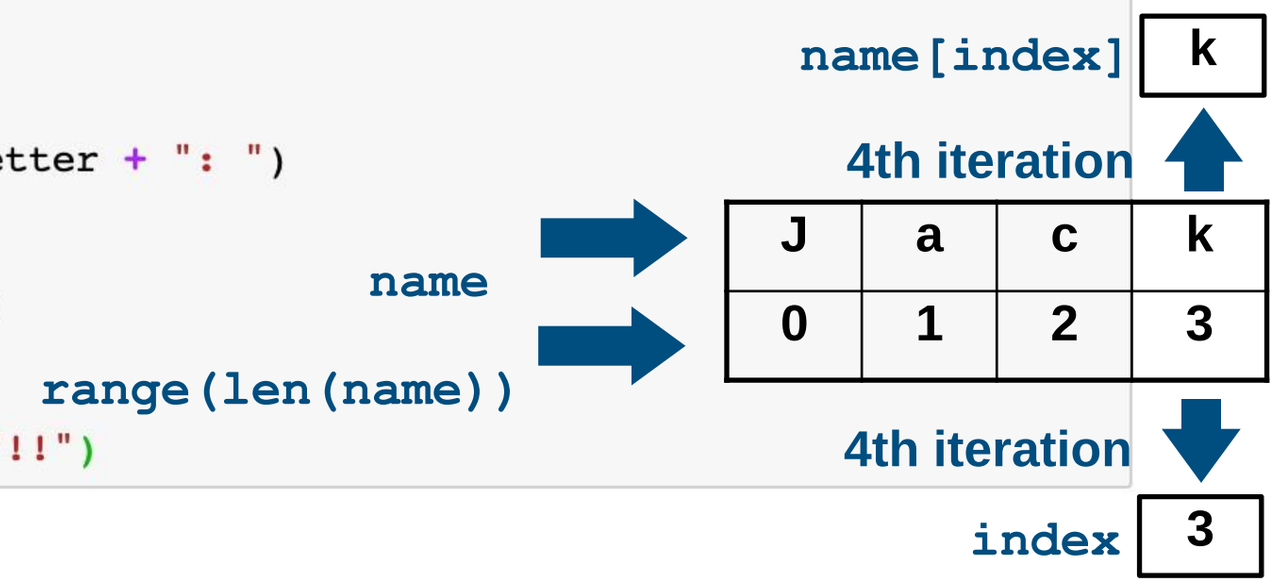
```
In [14]: name = input("What is your name? ")

for index in range(len(name)):  # Iterate integers 0, 1, ..., len(name)-1
    [redacted]
    if letter == ' ':
        continue

    print("Give me a " + letter + ": ")
    print(letter + "!!!")

print("What's that spell?")
print(name + "!!")

print("Go! Go! " + name + "!!")
```



Strings

- Strings are collections of characters
  - ▶ Iterate characters via [redacted]
    - ✓ Use [redacted] as the index of characters

```
In [14]: name = input("What is your name? ")

for index in range(len(name)):  # Iterate integers 0, 1, ..., len(name)-1
    [redacted]
    if letter == ' ':
        continue

    print("Give me a " + letter + ": ")
    print(letter + "!!!")

print("What's that spell?")
print(name + "!!")

print("Go! Go! " + name + "!!")
```



Strings

- Methods of strings
  - A method is a special function associated with an object
  - A method is called via the syntax

Strings

- Methods of strings
  - Case conversion

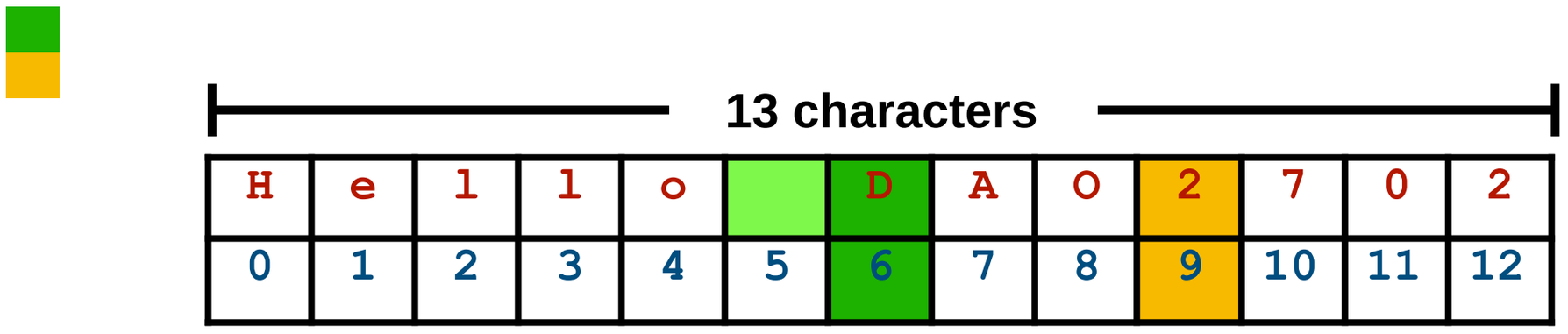
In [1]:
```
"""Case conversion"""

line = "all work and no play makes Jack a dull boy"

line_upper = line.upper()       # Convert all letters to upper case
line_lower = line.lower()       # Convert all letters to lower case
line_cap = line.capitalize()    # Convert the first letter to upper case
line_swap = line.swapcase()     # Swap upper and lower case
line_title = line.title()       # Capitalize the 1st letter of each word

print(line_upper)
print(line_lower)
print(line_cap)
print(line_swap)
print(line_title)
```

```
ALL WORK AND NO PLAY MAKES JACK A DULL BOY
all work and no play makes jack a dull boy
All work and no play makes jack a dull boy
ALL WORK AND NO PLAY MAKES jACK A DULL BOY
All Work And No Play Makes Jack A Dull Boy
```

Strings

- Methods of strings
  - Search items in a string

```
In [4]: """Find a character in a string"""

        greetings = "Hello DAO2702"

        print(greetings.find("DAO"))     # Print the index of "D"
        print(greetings.find("2"))       # Print the index of the first "2"
```

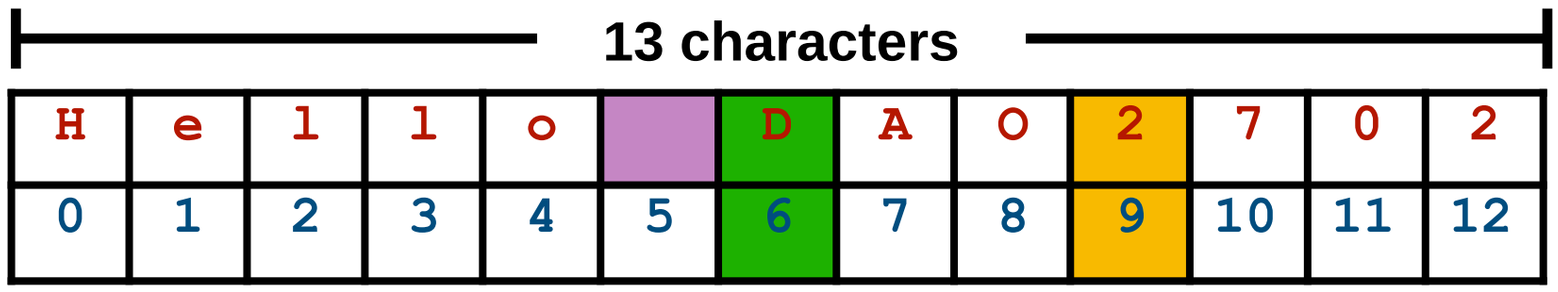


Strings

- Methods of strings
  - Search items in a string

```
In [4]: """Find a character in a string"""

        greetings = "Hello DAO2702"

        print(greetings.find("DAO"))     # Print the index of "D"
        print(greetings.find("2"))       # Print the index of the first "2"
```

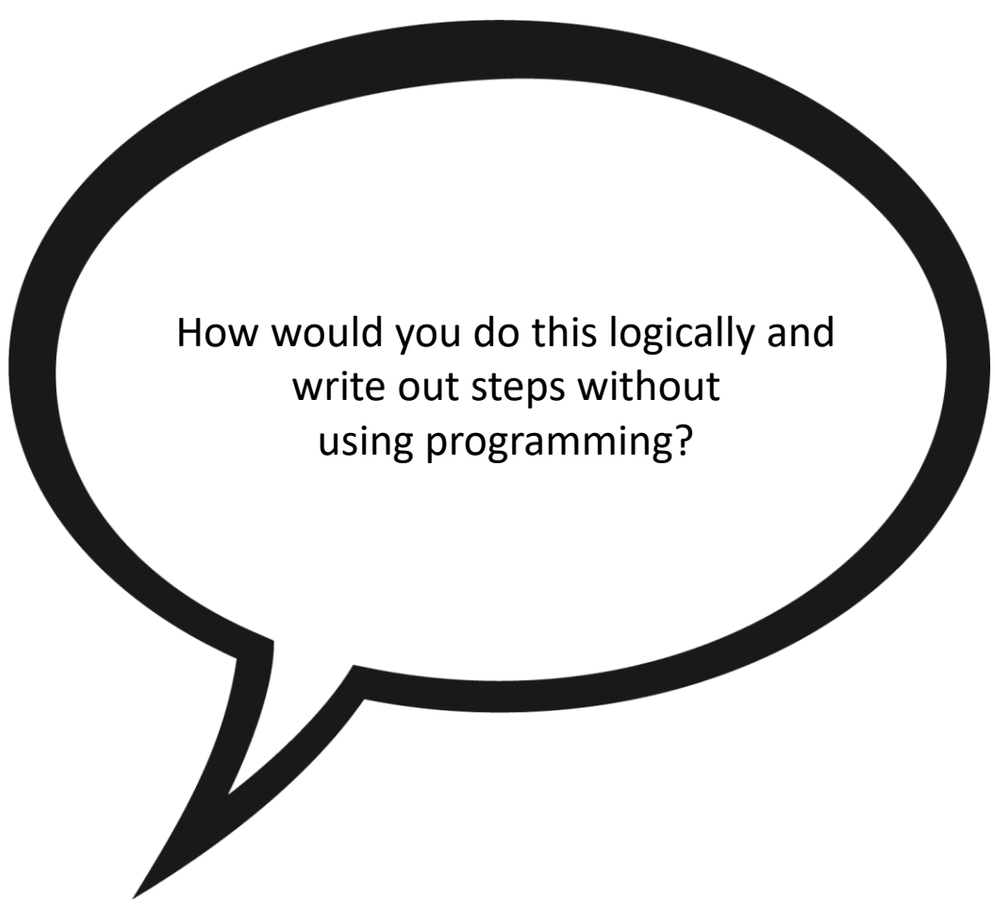
This method returns -1 if the given item cannot be found



Strings

Example 4

Write a program to print the initials of a name given by the user. For example, if the given name is "John Fitzgerald Kennedy", the printed out message is "JFK".



Strings

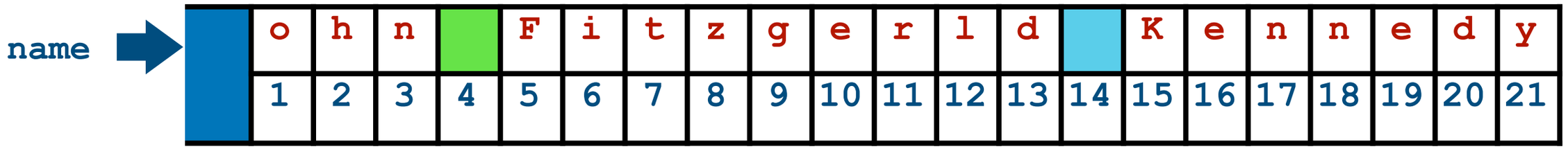
Example 4

Write a program to print the initials of a name given by the user. For example, if the given name is "John Fitzgerald Kennedy", the printed out message is "JFK".

In [17]:
```
name = input("Key in a name: ")

initials = name[0]                                  # Get the first name initial
while True:
    index = name.find(' ')                          # Position of the first space
    if index == -1:                                 # Break if no space found
        break
    initials = initials + name[index+1]             # Add the letter after space
    name = name[index+1:]                           # Get a subset of name

print(initials)
```

```
Key in a name: John Fitzgerald Kennedy
JFK
```

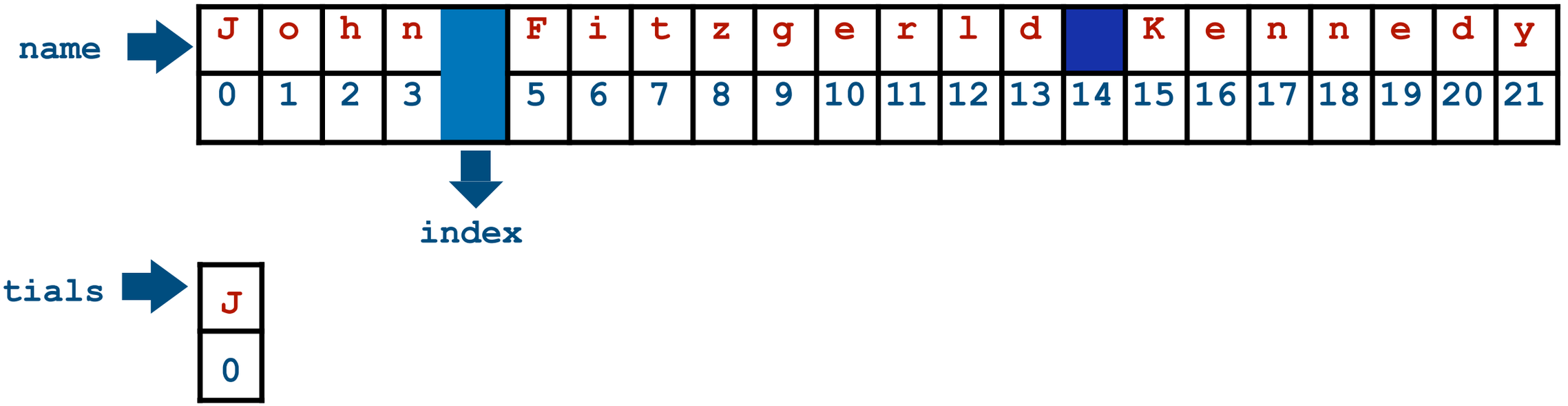
Strings




```
In [17]: name = input("Key in a name: ")

         initials = name[0]                              # Get the first name initial
         while True:
             index = name.find(' ')                      # Position of the first space
             if index == -1:
                 break                                   # Break if no space found
             initials = initials + name[index+1]         # Add the letter after space
             name = name[index+1:]                       # Get a subset of name

         print(initials)
```

```
Key in a name: John Fitzgerald Kennedy
JFK
```

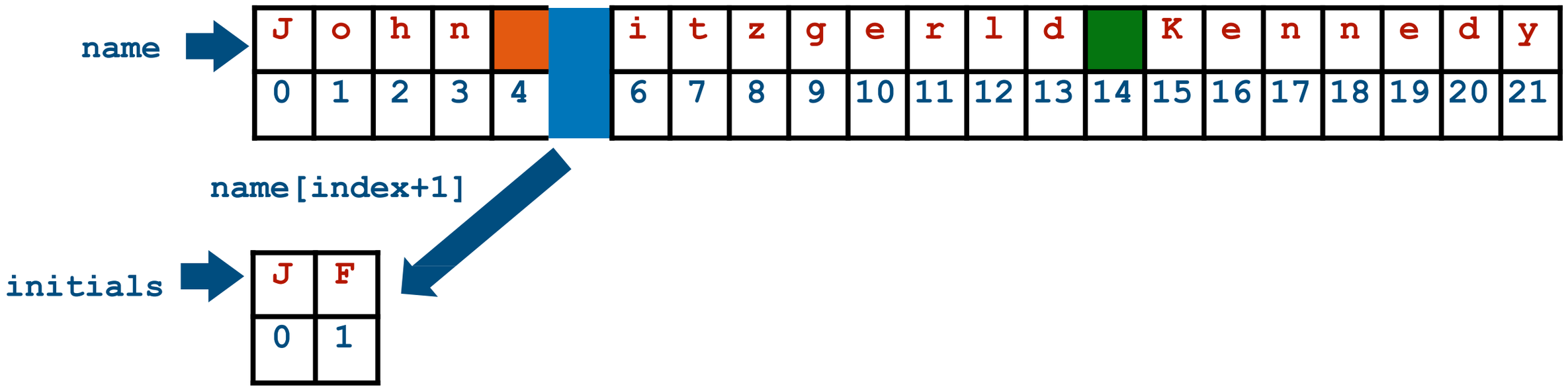
Strings



In [17]:
```
name = input("Key in a name: ")

initials = name[0]                              # Get the first name initial
while True:
    index = name.find(' ')                      # Position of the first space
    if index == -1:
        break                                   # Break if no space found
    initials = initials + name[index+1]         # Add the letter after space
    name = name[index+1:]                       # Get a subset of name

print(initials)
```

```
Key in a name: John Fitzgerald Kennedy
JFK
```

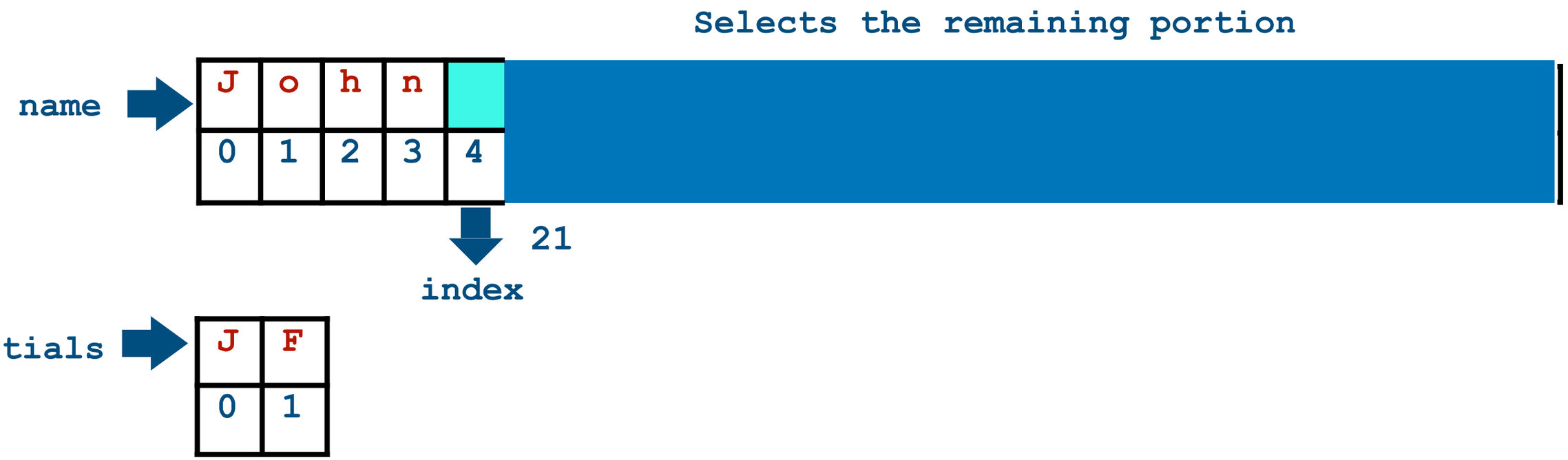
Strings



In [17]:
```
name = input("Key in a name: ")

initials = name[0]                          # Get the first name initial
while True:
    index = name.find(' ')                  # Position of the first space
    if index == -1:
        break                               # Break if no space found
    initials = initials + name[index+1]     # Add the letter after space
    name = name[index+1:]                   # Get a subset of name

print(initials)
```

```
Key in a name: John Fitzgerald Kennedy
JFK
```

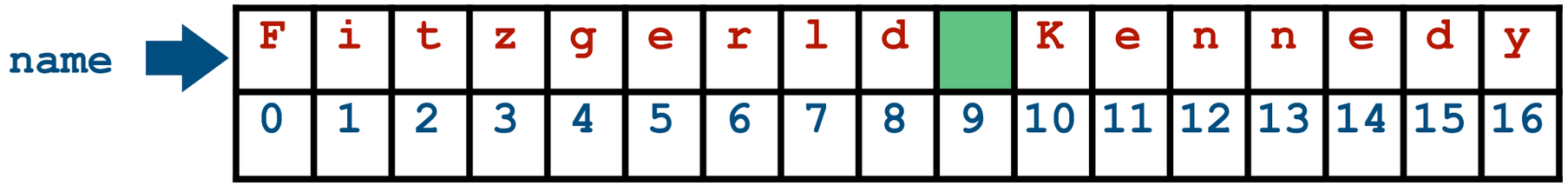
Strings

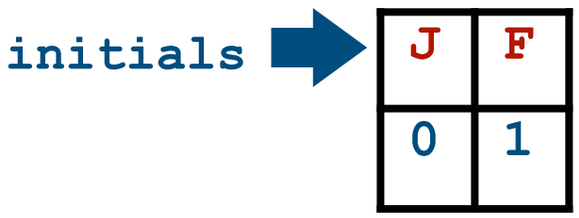


In [17]:
```
name = input("Key in a name: ")

initials = name[0]                              # Get the first name initial
while True:
    index = name.find(' ')                      # Position of the first space
    if index == -1:
        break                                   # Break if no space found
    initials = initials + name[index+1]         # Add the letter after space
    name = name[index+1:]                       # Get a subset of name

print(initials)
```

```
Key in a name: John Fitzgerald Kennedy
JFK
```

Strings

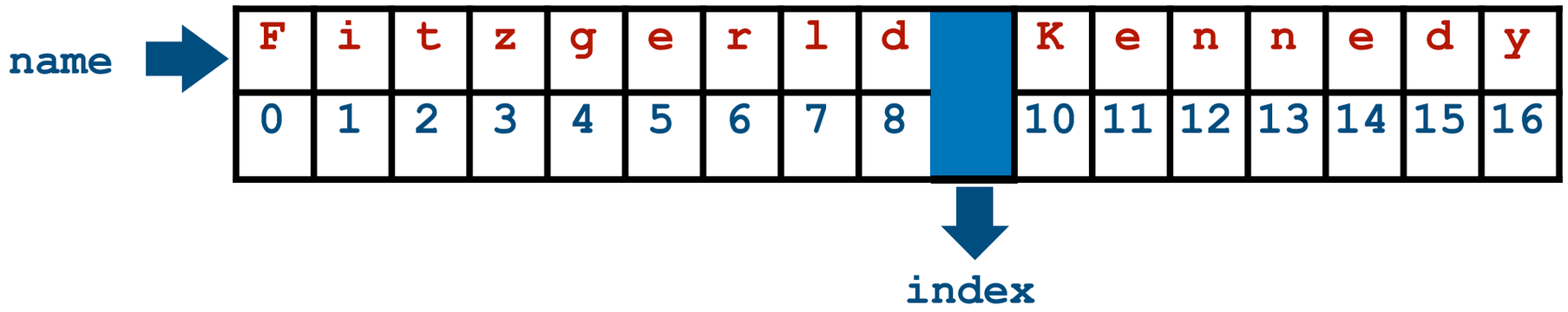


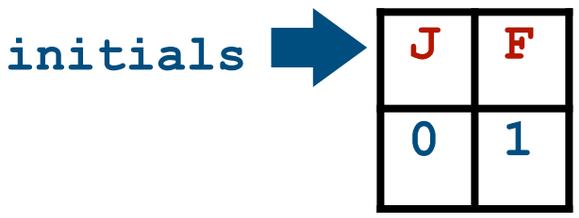


In [17]:
```
name = input("Key in a name: ")

initials = name[0]                              # Get the first name initial
while True:
    index = name.find(' ')                      # Position of the first space
    if index == -1:
        break                                   # Break if no space found
    initials = initials + name[index+1]         # Add the letter after space
    name = name[index+1:]                       # Get a subset of name

print(initials)
```

```
Key in a name: John Fitzgerald Kennedy
JFK
```

Strings

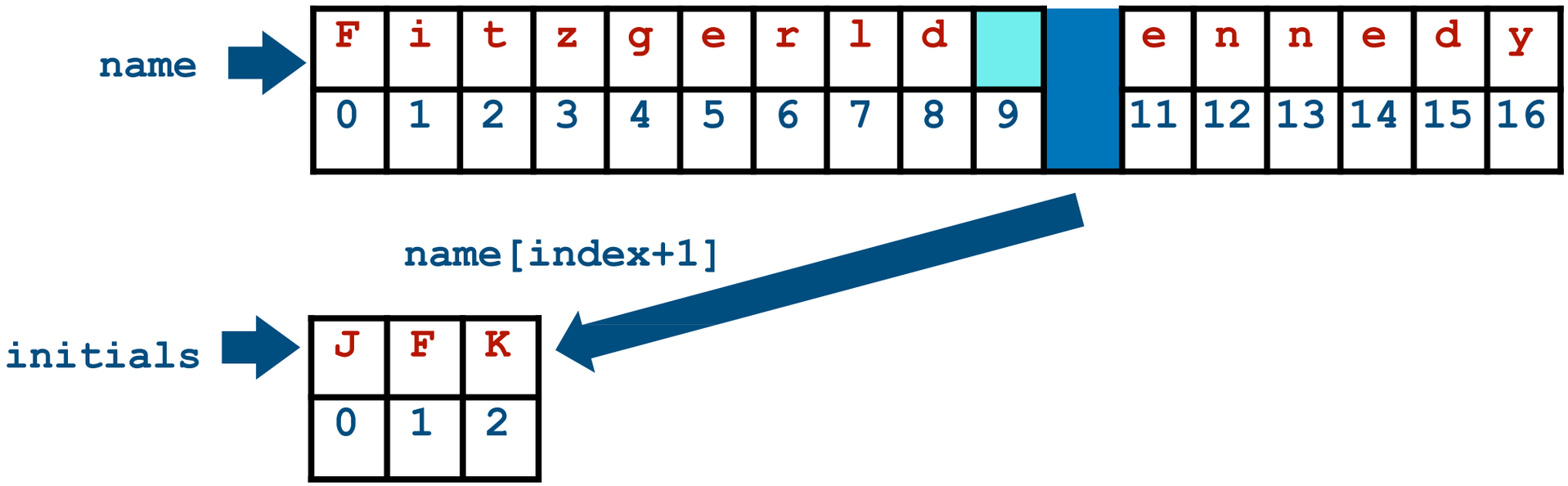




In [17]:
```
name = input("Key in a name: ")

initials = name[0]                              # Get the first name initial
while True:
    index = name.find(' ')                      # Position of the first space
    if index == -1:
        break                                   # Break if no space found
    initials = initials + name[index+1]         # Add the letter after space
    name = name[index+1:]                       # Get a subset of name

print(initials)
```

```
Key in a name: John Fitzgerald Kennedy
JFK
```

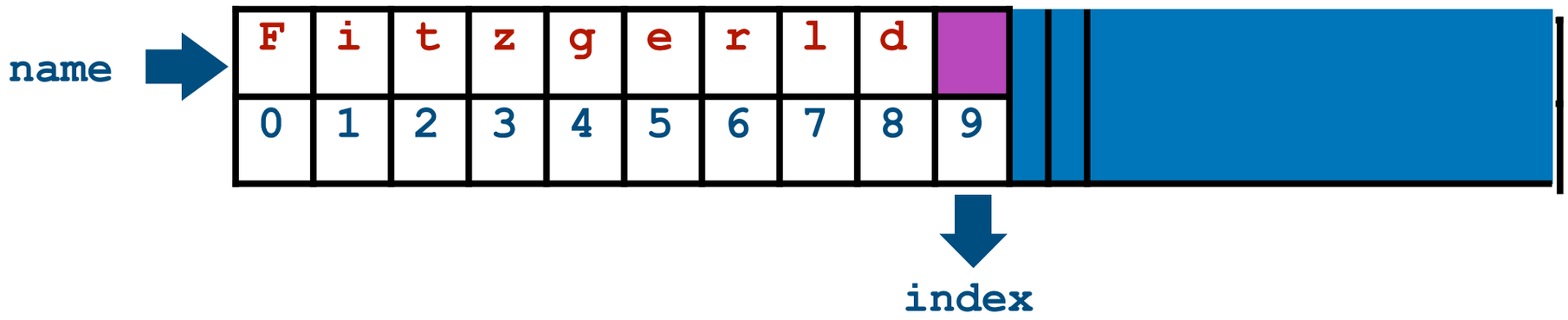
Strings

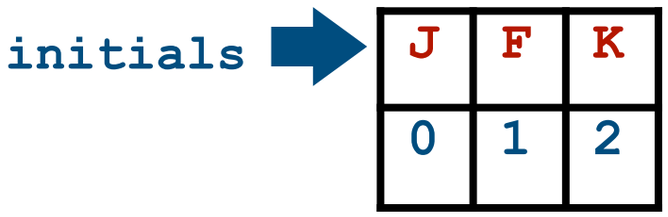



```
In [17]: name = input("Key in a name: ")

         initials = name[0]                          # Get the first name initial
         while True:
             index = name.find(' ')                  # Position of the first space
             if index == -1:
                 break                               # Break if no space found
             initials = initials + name[index+1]     # Add the letter after space
             name = name[index+1:]                   # Get a subset of name

         print(initials)
```

```
Key in a name: John Fitzgerald Kennedy
JFK
```

Strings



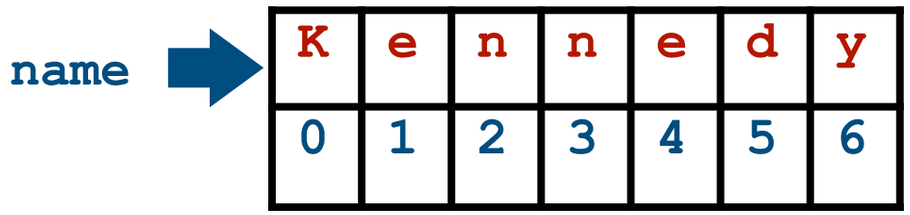


In [17]:
```
name = input("Key in a name: ")

initials = name[0]                              # Get the first name initial
while True:
    index = name.find(' ')                      # Position of the first space
    if index == -1:
        break                                   # Break if no space found
    initials = initials + name[index+1]         # Add the letter after space
    name = name[index+1:]                       # Get a subset of name

print(initials)
```

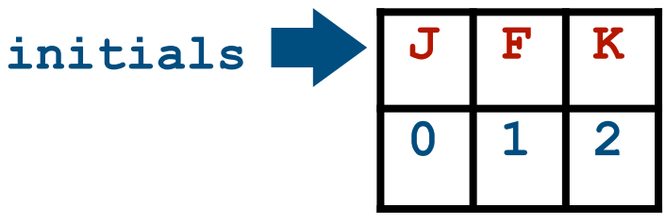
```
Key in a name: John Fitzgerald Kennedy
JFK
```

Strings

https://www.w3schools.com/python/ref_string_find.asp



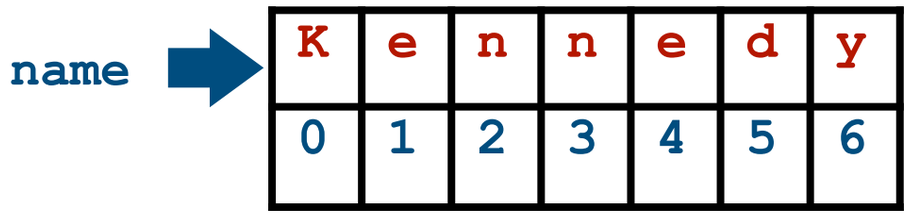
Cannot find a whitespace in `name`, so return -1



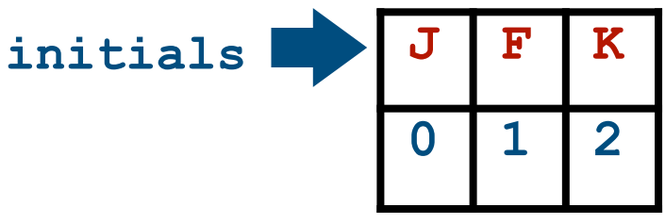
In [17]:
```
name = input("Key in a name: ")

initials = name[0]                              # Get the first name initial
while True:
    index = name.find(' ')                      # Position of the first space
    if index == -1:
        break                                   # Break if no space found
    initials = initials + name[index+1]         # Add the letter after space
    name = name[index+1:]                       # Get a subset of name

print(initials)
```

```
Key in a name: John Fitzgerald Kennedy
JFK
```

Strings



Cannot find a whitespace in `name`, so return -1



In [17]:
```
name = input("Key in a name: ")

initials = name[0]                          # Get the first name initial
while True:
    index = name.find(' ')                  # Position of the first space
    if index == -1:
        break                               # Break if no space found
    initials = initials + name[index+1]     # Add the letter after space
    name = name[index+1:]                   # Get a subset of name

print(initials)
```

```
Key in a name: John Fitzgerald Kennedy
JFK
```

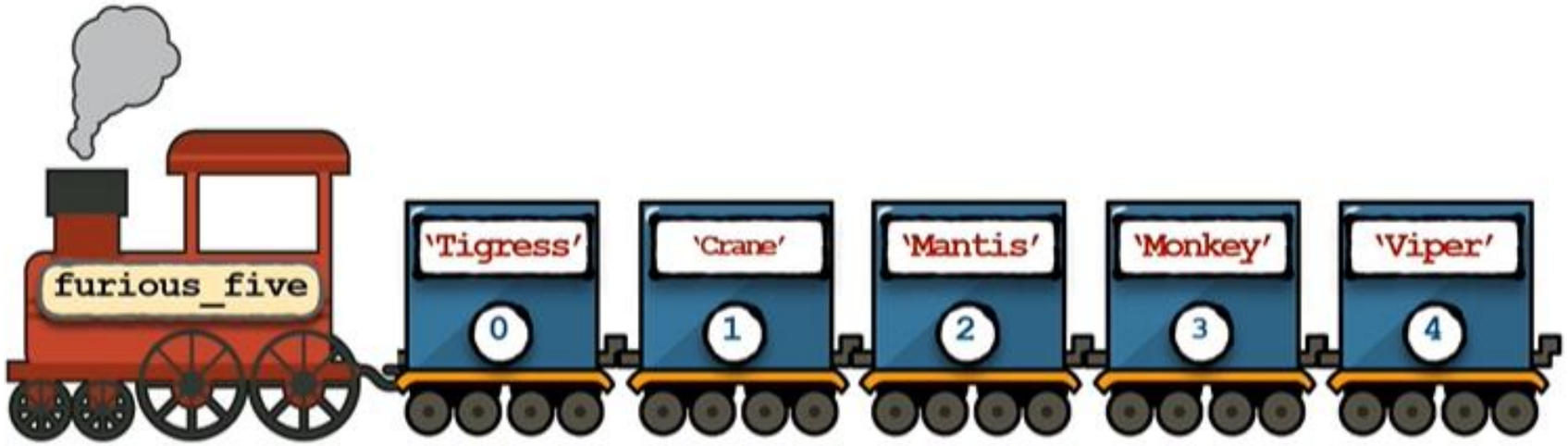
List

- List
  - a collection which is ordered and changeable. Allows duplicate members

```
# An empty list
I_feel_empty = [ ]

# A list with the same data types
furious_five = [ ‘ Tigress’ , ‘ Crane ’ , ‘ Mantis ’ , ‘ Monkey ’ , ‘ Viper ’ ]

# A list of mixed data types
my_answers = [ ‘ B ’ , ‘ C ’ ,          # Answers for MCQs
               False , True ,          # Answers for True or False questions
               0.256 ]                 # Answers for a quantitative question
```

Similarities between list and strings

- Similarity: Indexing and slicing



List

['Tigress' , 'Crane' , 'Mantis' , 'Monkey' , 'Viper']

'Tigress'	'Crane'	'Mantis'	'Monkey'	'Viper'
0	1	2	3	4
-5	-4	-3	-2	-1

String

"Hello"

H	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

Differences between list and strings

- Mutability: element-wise segment

```
In [7]: my_answers = ['B', 'C', False, True, 0.256, 2]
        print(my_answers)                    # Print the original list

        my_answers[1] = 'D'                  # Modify the 2nd item in the list
        print(my_answers)                    # Print the modified list

        my_answers[2:4] = [True, False]      # Modify the 3rd and the 4th item
        print(my_answers)                    # Print the modified list
```

```
['B', 'C', False, True, 0.256, 2]
['B', 'D', False, True, 0.256, 2]
['B', 'D', True, False, 0.256, 2]
```

If you try to change some characters in a string via indexing and slicing, an error message will be given!

List methods

- List
  - Append, extend a list

the_list									
	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8

In [20]:
```
the_list = [1, 2, 3, 4]
print(the_list)

the_list.append(5)                  # Item 5 is added to the list
print(the_list)

another_list = [6, 7, 8, 9]
the_list.extend(another_list)       # Another list is added to the list
print(the_list)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

extend the list by adding [6,7,8,9] to the end

List methods

- List
  - Insert an element to a list

the_list

1	2	“here”	3	4
0	1	2	3	4

In [23]:
```
the_list = [1, 2, 3, 4]
the_list.insert(2, 'here')   # Insert an item, with the index to be 2

print(the_list)
```

```
[1, 2, 'here', 3, 4]
```

List methods

- List
  - Remove an element from the list using remove()

the_list

1	2	“here”	3	4
0	1	2	3	4

```
In [23]: the_list = [1, 2, 'here', 3, 4]

         the_list.remove('here')        # Remove "here" from the list
         print(the_list)
```

List methods

- List
  - Remove an element from the list using pop()

the_list

1	2	3	4	“last item ”
0	1	2	3	4

```
In [24]: the_list = [1, 2, 'here', 3, 4, 'last item']

         pop_item = the_list.pop(2)  # Remove the 3rd item, and return this item
         print(pop_item)
         print(the_list)

         here
         [1, 2, 3, 4, 'last item']
```

List methods

- List
  - Remove an element from the list using pop()

the_list

1	2	“here”	3	4
0	1	2	3	4

```
In [25]: the_list = [1, 2, 'here', 3, 4, 'last item']

         pop_item = the_list.pop()    # The default value of pop index is -1
         print(pop_item)
         print(the_list)

         last item
         [1, 2, 'here', 3, 4]
```

Dictionary

- Dictionary

Example

Print the summary of personal information stored in a list.

Name: Jack Sparrow
Age: 30
Workplace: Black Pearl
Title: Captain

Dictionary

- Dictionary

Example

Print the summary of personal information stored in a list.

Name: Jack Sparrow
Age: 30
Workplace: Black Pearl
Title: Captain

```
In [39]: personal_info = ['Jack Sparrow', 30, 'Black Pearl', 'Captain']

         print('Name: ' + personal_info[0])
         print('Age: ' + str(personal_info[1]))
         print('Workplace: ' + personal_info[2])
         print('Title: ' + personal_info[3])
```

```
Name: Jack Sparrow
Age: 30
Workplace: Black Pearl
Title: Captain
```

Is this a good method?

Dictionary

- Dictionary

Example

Print the summary of personal information stored in a list.

Name: Jack Sparrow
Age: 30
Workplace: Black Pearl
Title: Captain

An “elegant” and efficient solution via Python dictionaries

```
In [42]: personal_info = {'name': 'Jack Sparrow',
                         'age': 30,
                         'workplace': 'Black Pearl',
                         'title': 'Captain'}

         print('Name: ' + personal_info['name'])
         print('Age: ' + str(personal_info['age']))
         print('Workplace: ' + personal_info['workplace'])
         print('Title: ' + personal_info['title'])
```

```
Name: Jack Sparrow
Age: 30
Workplace: Black Pearl
Title: Captain
```

Dictionary

- Iterate a Dictionary

```
In [44]: personal_info = {'name': 'Jack Sparrow',
                         'age': 30,
                         'workplace': 'Black Pearl',
                         'title': 'Captain'}

for key in personal_info:                # Iterate the keys of the dictionary
    value = personal_info[key]           # Access the values
    print(key.title() + ': ' + str(value))
```

```
Name: Jack Sparrow
Age: 30
Workplace: Black Pearl
Title: Captain
```

‘Jack Sparrow’	30	‘Black Pearl’	‘Captain’
‘name’	‘age’	‘workplace’	‘title’

Roadmap

1. Introduction
2. Objects, Variables, and Data Types
3. Strings
4. **Control Flow: Boolean Type**

Control Flow: Boolean Type

- The Boolean Type Expressions
  - Status: [redacted] or [redacted]
  - Comparison operators

Operators	Remarks	Example
[redacted]	Equal	`x == y`
[redacted]	Not equal	`x != y`
[redacted]	Greater than or equal to	`x >= y`
[redacted]	Smaller than or equal to	`x <= y`
[redacted]	Greater than	`x > y`
[redacted]	Smaller than	`x < y`

Control Flow: Boolean Type

- The Boolean Type Expressions
  - Status: [redacted] or [redacted]
  - Comparison operators

In [19]:
```
print(2 <= 3)       # 2 <= 3 is true
print(3.5 > 4)      # 3.5 > 4 is false
print(2 == 2.0)     # 2 == 2.0 is true
print(2 != 2.0)     # 2 != 2.0 is false
```

```
True
False
True
False
```

Control Flow: Boolean Type

- The Boolean Type Expressions
  - Status: [redacted] or [redacted]
  - Identity operators

Operators	Remarks	Examples
[redacted]	Returns True if both variables are the same object	`x is y`
[redacted]	Returns True if two variables are not the same object	`x is not y`

Control Flow: Boolean Type

- The Boolean Type Expressions
  - Status: [redacted] or [redacted]
  - Identity operators

Operators	Remarks	Examples
[redacted]	Returns True if both variables are the same object	`x is y`
[redacted]	Returns True if two variables are not the same object	`x is not y`

```
In [19]: print(0 == 0.0)      # True because 0 and 0.0 have equal values
         print(0 is 0.0)      # False because they are not the same object
```

```
True
False
```

Control Flow: Boolean Type

- The Boolean Type Expressions
  - Status: [redacted] or [redacted]
  - Membership operators

Operators	Remarks	Examples
[redacted]	Returns True if it finds variable in the specified sequence and False otherwise	`x in y`
[redacted]	Returns False if it finds variable in the specified sequence and True otherwise	`x not in y`

Control Flow: Boolean Type

- The Boolean Type Expressions
  - Status: [redacted] or [redacted]
  - Membership operators

In [20]:
```
sentence = "All work and no play makes Jack a dull boy."

print('work' in sentence)      # True as "work" is in the sentence
print('Work' in sentence)      # False as "Work" is not in the sentence

print('John' not in sentence)  # True as "John" is not in the sentence
```

```
True
False
True
```

Control Flow: Boolean Type

- The Boolean Type Expressions
  - Status: [redacted] or [redacted]
  - Logic operators

Operators	Remarks	Examples
[redacted]	Returns True if both statements are True	`x >= 1 and y <= 1`
[redacted]	Returns True if either one statements is True	`x > 1 or y < 1`
[redacted]	Reverse the result, returns False if the result	`not x == 0`

Control Flow: Boolean Type

- The Boolean Type Expressions
  - Status: [redacted] or [redacted]
  - Logic operators

In [29]:
```
is_monday = True
is_public_holiday = False
is_weekends = False

no_school = is_public_holiday or is_weekends
have_dao2702 = is_monday and not is_public_holiday

print(no_school)
print(have_dao2702)
```

```
False
True
```

Control Flow: Boolean Type

- The Boolean Type Expressions
  - Status: [redacted] or [redacted]
  - Logic operators

In [29]:
```
is_monday = True
is_public_holiday = False
is_weekends = False

no_school = [redacted] or [redacted]
have_dao2702 = is_monday and not is_public_holiday

print(no_school)
print(have_dao2702)
```

The condition is True if at least one statement is True

```
[redacted]
True
```

Control Flow: Boolean Type

- The Boolean Type Expressions
  - Status: [redacted] or [redacted]
  - Logic operators

In [29]:
```
is_monday = True
is_public_holiday = False
is_weekends = False

no_school = is_public_holiday or is_weekends
have_dao2702 = [redacted] and [redacted]

print(no_school)
print(have_dao2702)
```

The condition is True if both statements are True

```
False
[redacted]
```

Roadmap

1. Introduction
2. Objects, Variables, and Data Types
3. Strings
4. **Control Flow: if**

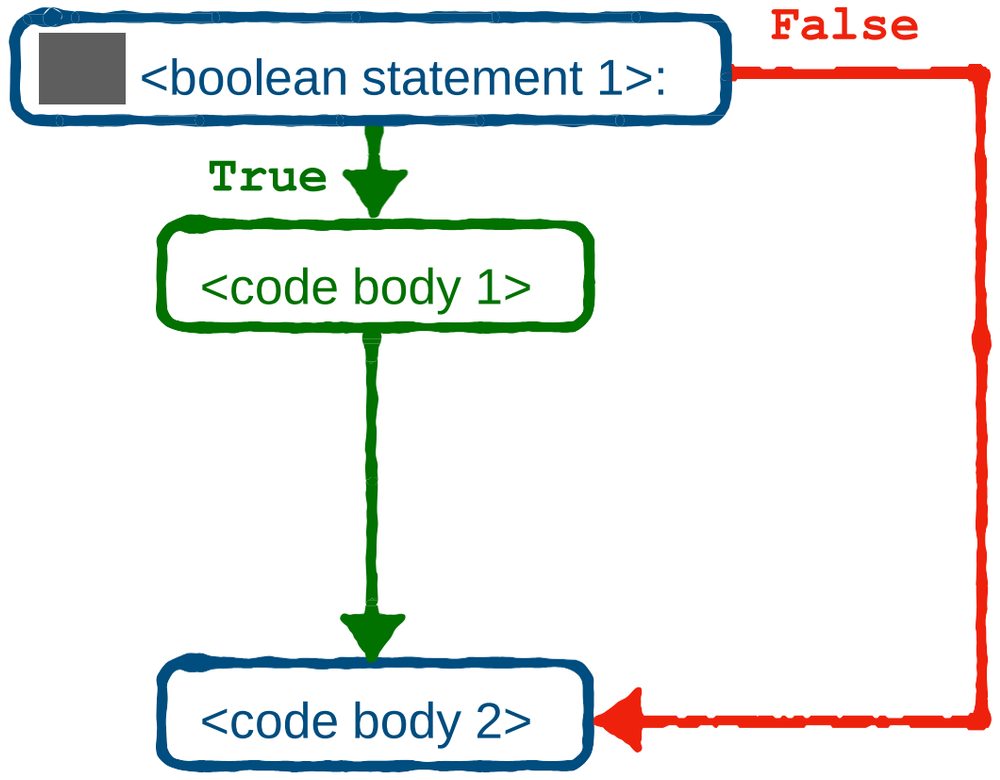
Control Flow: from Boolean Type…

- Conditional Statements
  - The [redacted] syntax enables different actions according to the status of given boolean expressions.

Control Flow: if

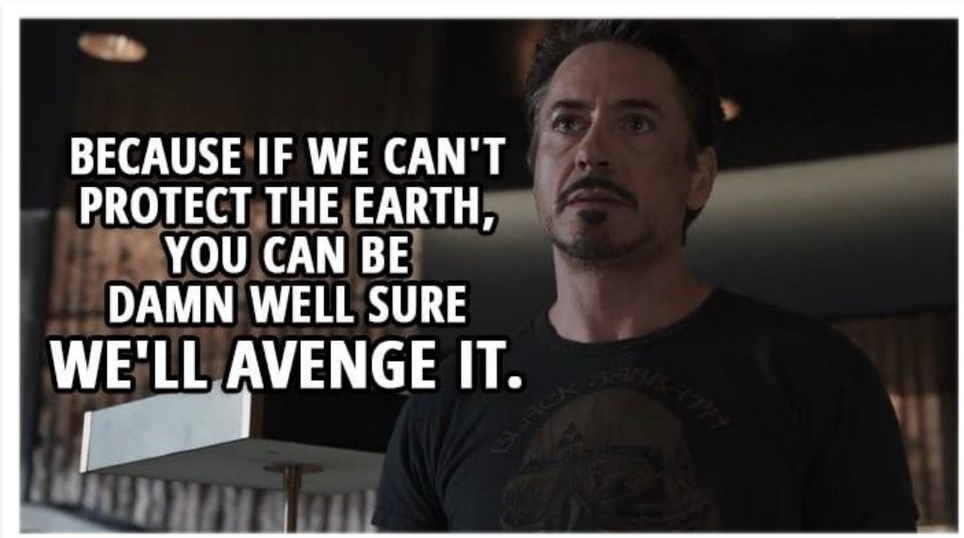
- Conditional Statements
  - ► The [redacted] syntax
    - ✓ Keyword [redacted]
      ✓ Execute <code body 1> if <boolean statement 1> is [redacted] and skip it if otherwise
      ,
    - ✓ Indent (**a tab or four spaces**) <code body 1> to differentiate it from <code body 2>

```
[redacted] <boolean statement 1>:
    <code body 1>
<code body 2>
```



Control Flow: if

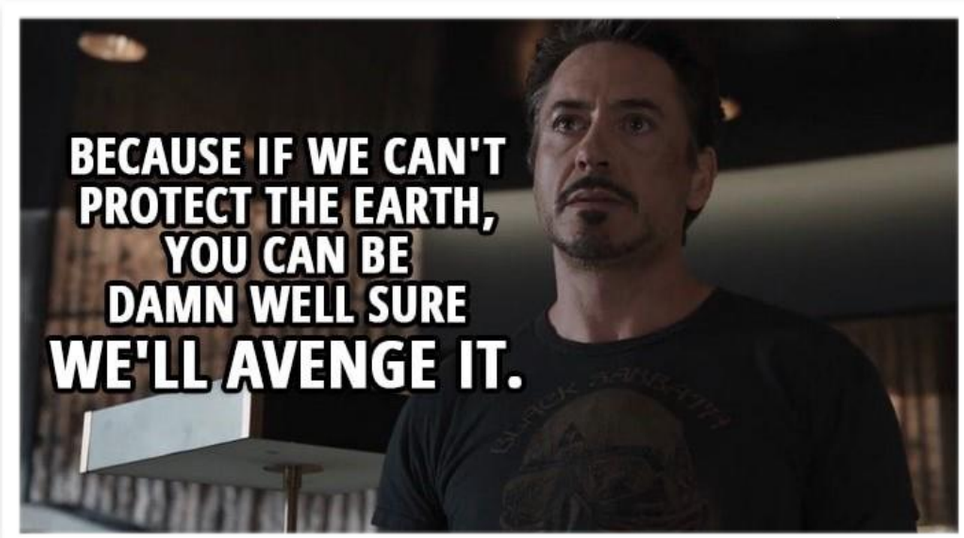
- Conditional Statements
  - The [redacted] syntax




```
can_protect_earth = False

if      can_protect_earth:
    print("We'll avenge it!")
print("Avengers assemble!")
```

Control Flow: if

- Conditional Statements
  - The [redacted] syntax

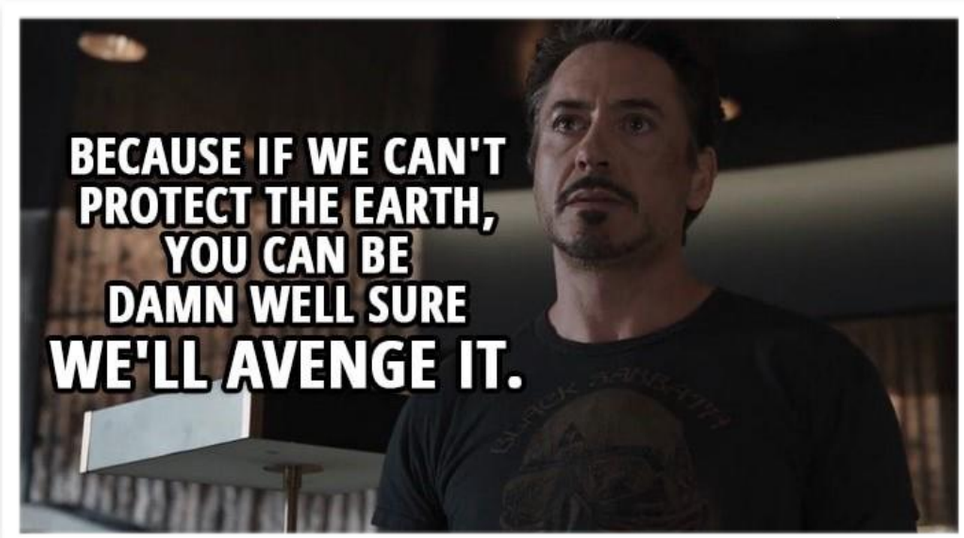



```
can_protect_earth = False

if      can_protect_earth:
    print("We'll avenge it!")
print("Avengers assemble!")
```

```
We'll avenge it!
Avengers assemble!
```

Control Flow: if

- Conditional Statements
  - The [redacted] syntax



Keyword

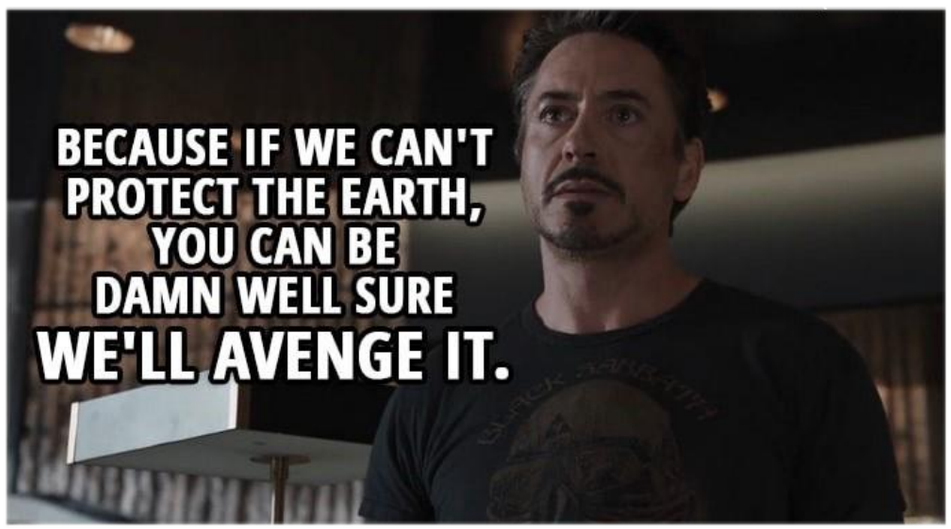
Do not forget the colon

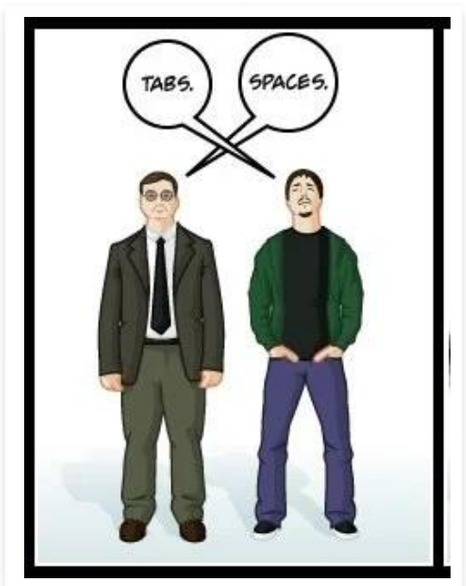
```
can_protect_earth = False

[redacted]     can_protect_earth[redacted]
    print("We'll avenge it!")
print("Avengers assemble!")
```

```
We'll avenge it!
Avengers assemble!
```

Control Flow: if

- Conditional Statements
  - The [redacted] syntax





Indentation (a tab or four spaces)

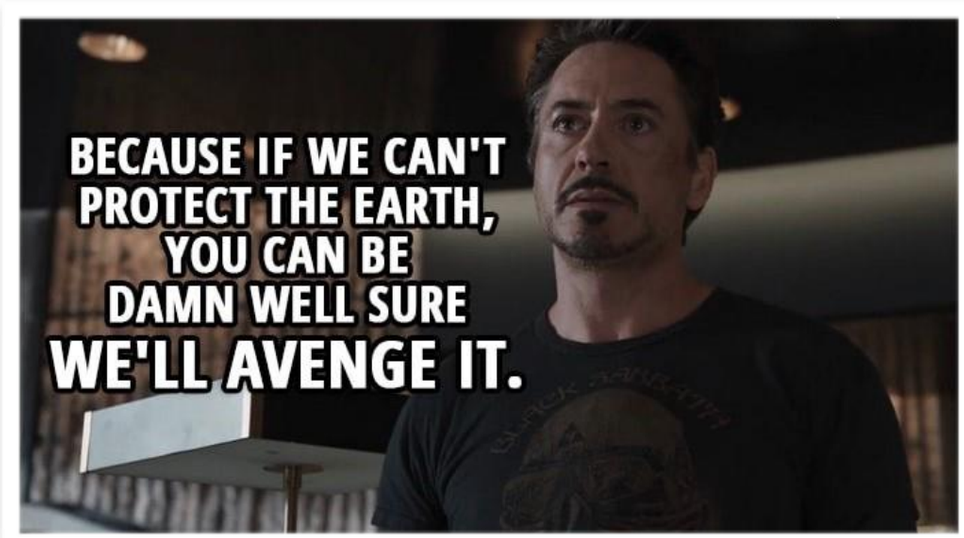
Execute this line if the condition is True

```
can_protect_earth = False

if      can_protect_earth:
    [redacted]
print("Avengers assemble!")
```

```
We'll avenge it!
Avengers assemble!
```

Control Flow: if

- Conditional Statements
  - The [redacted] syntax



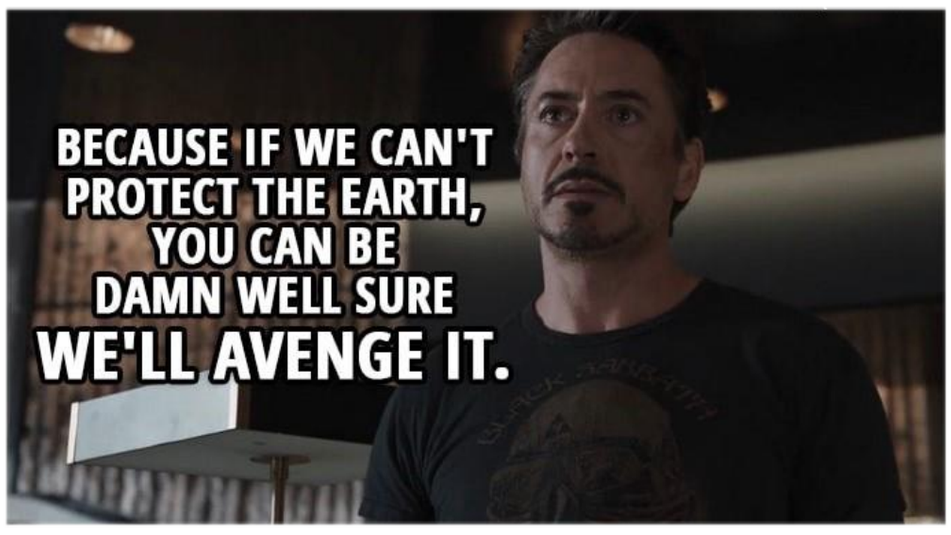
This line is out of the conditional statement

```
can_protect_earth = False

if      can_protect_earth:
    print("We'll avenge it!")
[redacted]
```

```
We'll avenge it!
Avengers assemble!
```

Control Flow: if

- Conditional Statements
  - ► The [redacted] syntax
    - ✓ What is the printed message if [redacted] is [redacted]



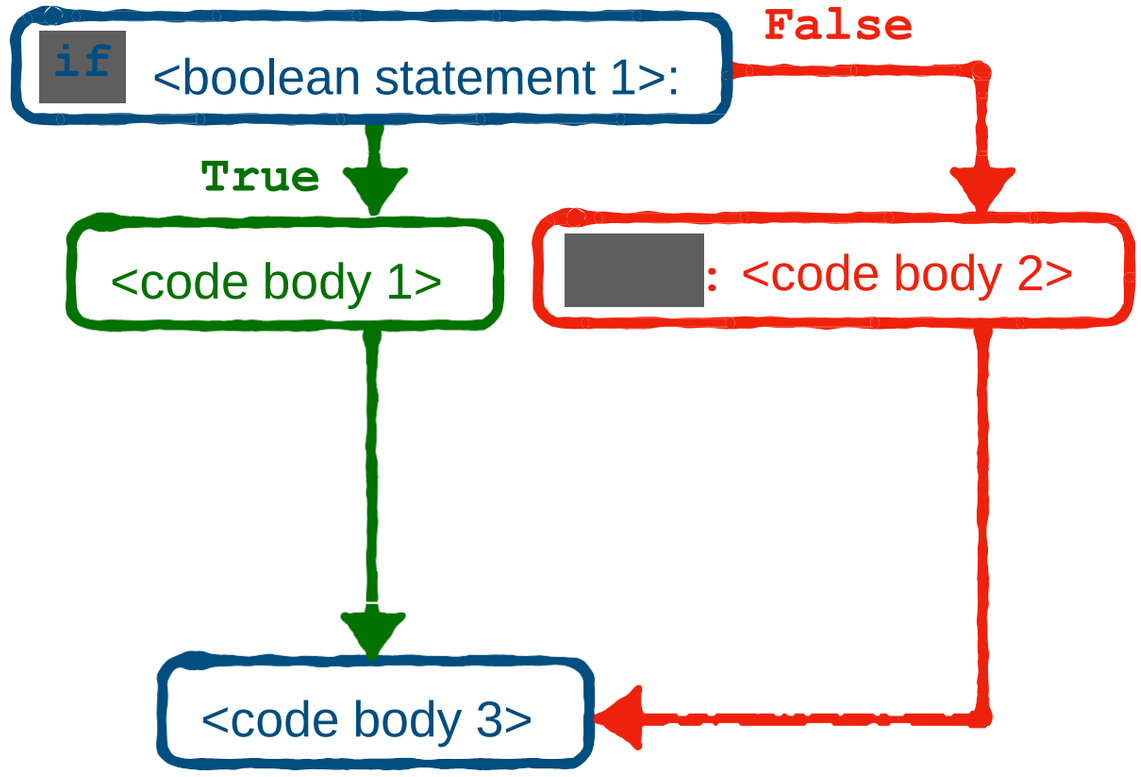

```
can_protect_earth = False

if      can_protect_earth:
    print("We'll avenge it!")
print("Avengers assemble!")
```

```
We'll avenge it!
Avengers assemble!
```

Control Flow: if-else

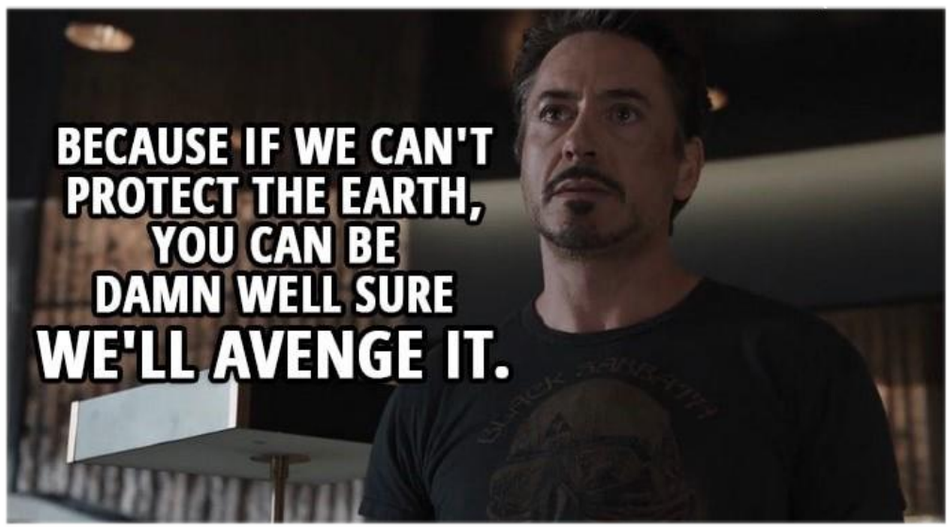
- Conditional Statements
  - ► The [redacted] syntax
    - ✓ Keyword [redacted] and [redacted]
    - ✓ Execute <code body 1> if <boolean statement 1> is [redacted]
    - ✓ Execute <code body 2> if <boolean statement 1> is [redacted]
    - ✓ Same indentations for [redacted]

```
[redacted] <boolean statement 1>:
    <code body 1>
[redacted]:
    <code body 2>
<code body 3>
```



Control Flow: if-else

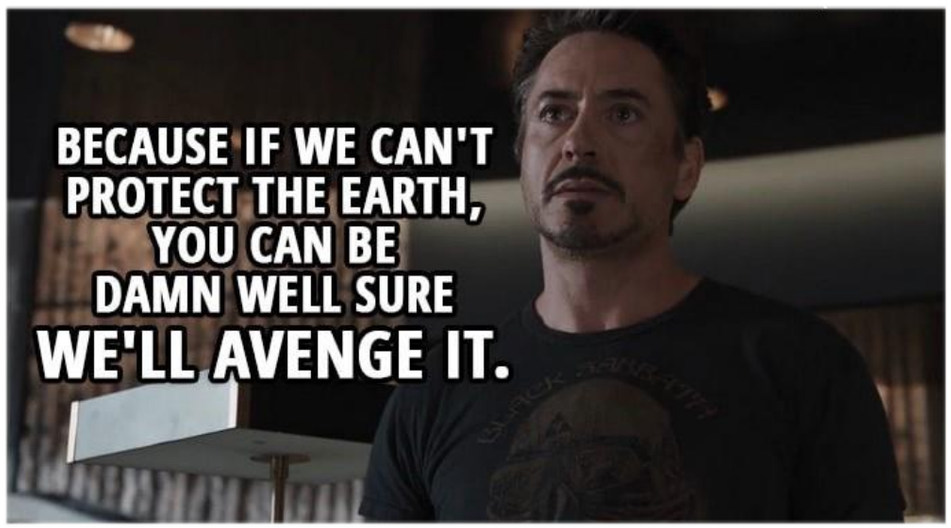
- Conditional Statements
  - The [redacted] syntax




```
can_protect_earth = True

if      can_protect_earth:
    print("We'll avenge it!")
else:
    print("We'll celebrate it!")
print("Avengers assemble!")
```

Control Flow: if-else

- Conditional Statements
  - The [redacted] syntax

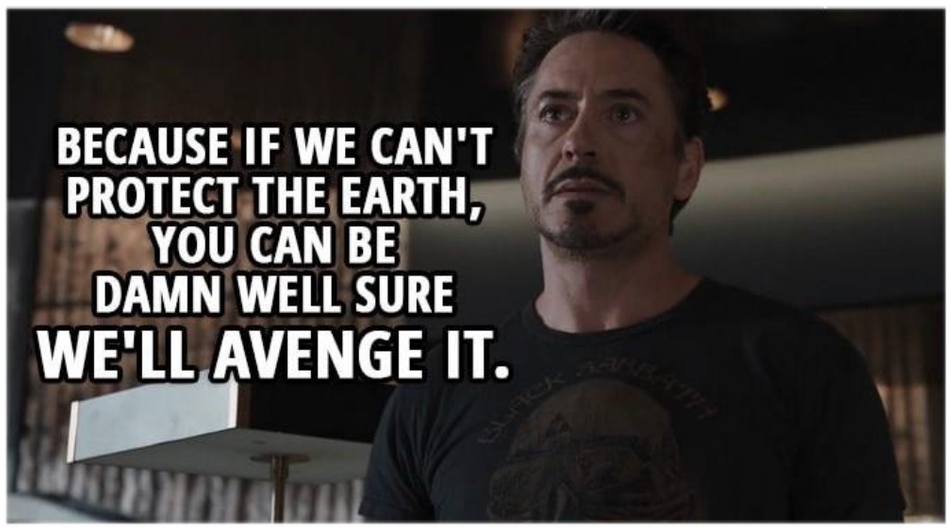



```
can_protect_earth = True

if      can_protect_earth:
    print("We'll avenge it!")
else:
    print("We'll celebrate it!")
print("Avengers assemble!")
```

```
We'll celebrate it!
Avengers assemble!
```

Control Flow: if-else

- Conditional Statements
  - The [redacted] syntax




```
can_protect_earth = True

if      can_protect_earth:
    print("We'll avenge it!")
else:
    [redacted]
print("Avengers assemble!")
```

Indentation (a tab or four spaces)

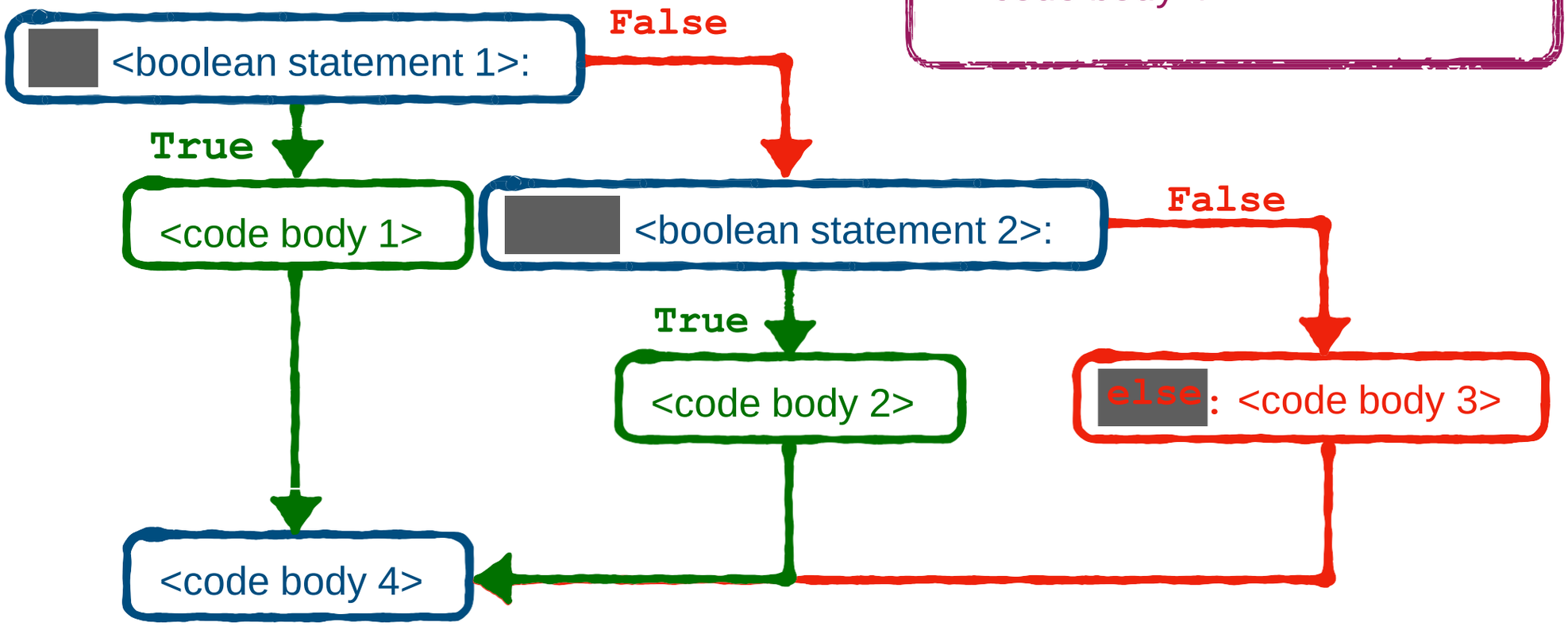
Execute this line if the condition is False

```
We'll celebrate it!
Avengers assemble!
```

Control Flow: if-elif-else

- Conditional Statements
  - ► Th [redacted] syntax
    e
    ✓ Keyword elif to add more conditions

```
[redacted] <boolean statement 1>:
    <code body 1>
[redacted]: <boolean statement 2>:
    <code body 2>
[redacted]:
    <code body 3>
<code body 4>
```



Control Flow: if

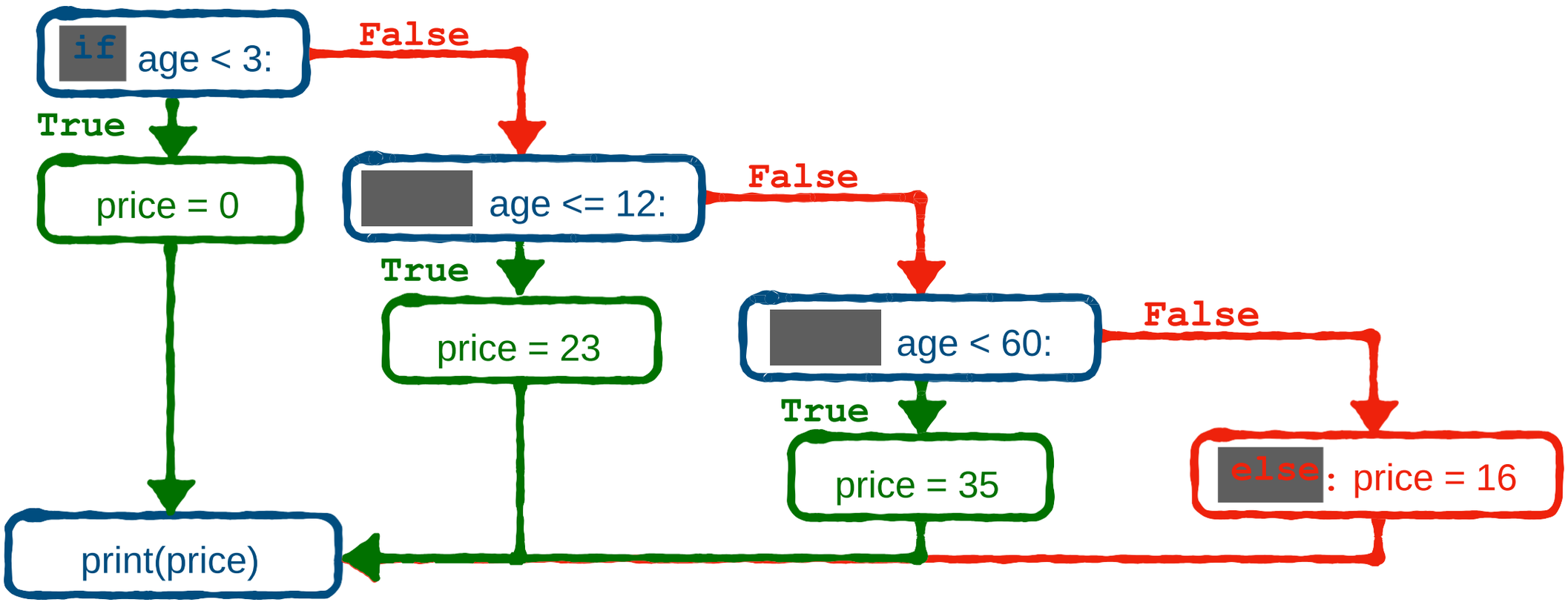
- Conditional Statements

Example 4

The price of ticket for Disneyland is $35 for adults, and $16 for senior citizens (60 years old or above). The ticket price for children aged 3 to 12 years old, is $23, and it is free for children under 3 years old. Write a program to determine the ticket price given a visitor's age.

Control Flow: if

- Conditional Statements



Control Flow: if

- Conditional Statements

Example 4

The price of ticket for Disneyland is $35 for adults, and $16 for senior citizens (60 years old or above). The ticket price for children aged 3 to 12 years old, is $23, and it is free for children under 3 years old. Write a program to determine the ticket price given a visitor's age.

In [32]:
```
age = 20

if age < 3:
    ticket_price = 0
elif age <= 12:
    ticket_price = 23
elif age < 60:
    ticket_price = 35
else:
    ticket_price = 16

print('The ticket price is: $' + str(ticket_price))
```

```
The ticket price is: $35
```

Control Flow: if

- Conditional Statements

Example 5

A news boy ordered 550 newspapers to sell. Each piece of newspaper costs 10 cents and can be sold at 60 cents. Write a program to calculate the total profit, given a fixed demand.

In [24]:
```
cost = 0.1
price = 0.6
order = 550

demand = 300

if demand < order:
    sold = demand
else:
    sold = order
profit = price*sold - cost*order

print("Newspapers sold: " + str(sold))
print("Total profit: " + str(profit))
```

Control Flow: if

- Conditional Statements

Example 5

A news boy ordered 550 newspapers to sell. Each piece of newspaper costs 10 cents and can be sold at 60 cents. Write a program to calculate the total profit, given a fixed demand.

In [25]:
```
cost = 0.1
price = 0.6
order = 550

demand = 300

sold = demand if demand < order else order

profit = price*sold - cost*order

print("Newspapers sold: " + str(sold))
print("Total profit: " + str(profit))
```

```
Newspapers sold: 300
Total profit: 125.0
```

Control Flow: if

- Conditional Statements

Example 5

A news boy ordered 550 newspapers to sell. Each piece of newspaper costs 10 cents and can be sold at 60 cents. Write a program to calculate the total profit, given a fixed demand.

In [25]:
```
cost = 0.1
price = 0.6
order = 550

demand = 300

sold = min(order, demand)

profit = price*sold - cost*order

print("Newspapers sold: " + str(sold))
print("Total profit: " + str(profit))
```

```
Newspapers sold: 300
Total profit: 125.0
```

Coding Style

Zen of Python:

- **Readability counts**

Control Flow: if

- Conditional Statements

Question

What is the output message of the following code?

```
x = 2.5
y = 3
z = 5 if x >= 3 or (not y <= 3) else 1

print(z)
```

A. 5

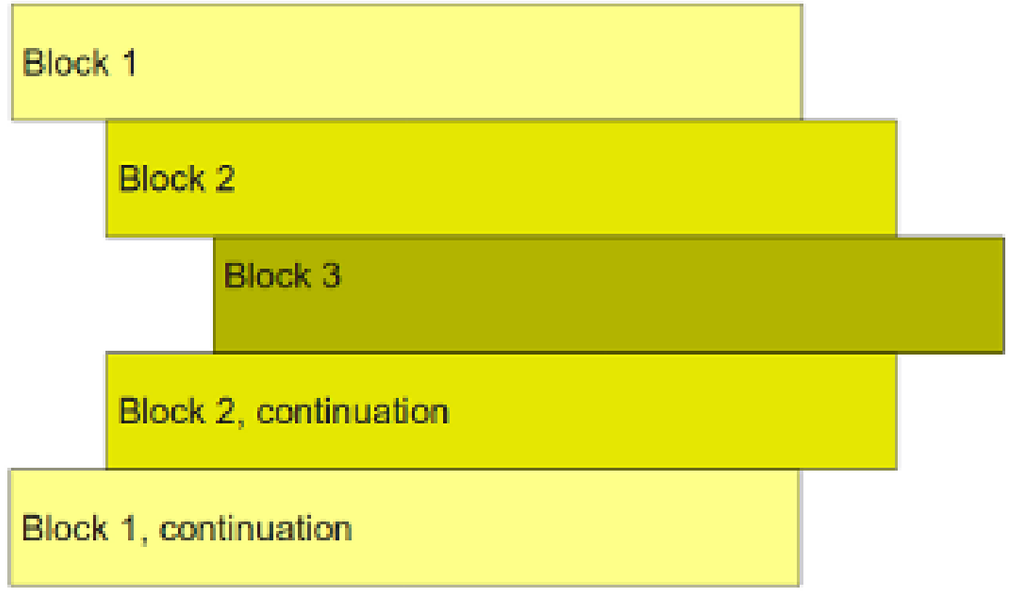
B. True

C. False

D. 1

Control Flow: if

- Conditional Statements
  - The nested **`if-elif-else`** structure



Control Flow: if

- Conditional Statements
  - The nested **`if-elif-else`** structure

Example 5

A news boy ordered 550 newspapers to sell. Each piece of newspaper costs 10 cents and can be sold at 60 cents. Every day, up to 120 pieces of newspaper can be recycled at a price of 5 cents. Write a program to calculate the total profit, given the demand quantity to be 300 and the order quantity to be 450.

Control Flow: if

- Conditional Statements
  - The nested **`if-elif-else`** structure

```
1  cost = 0.1
2  price = 0.6
3  rec_price = 0.05
4
5  order = 450
6  demand = 300
7
8  if demand < order:
9      sold = demand
10     left = order - sold
11     if left > 120:
12         recycle = 120
13     else:
14         recycle = left
15 else:
16     sold = order
17
18 profit = price*sold + rec_price*recycle - cost*order
19
20 print("Newspapers sold: " + str(sold))
21 print("Newspapers recycled: " + str(recycle))
22 print("Total profit: " + str(profit))
```

Inner branching structures

Leftover only occur in the case that demand is lesser than order

Roadmap

1. Introduction
2. Objects, Variables, and Data Types
3. Strings
4. **Control Flow: while**

Control Flow: while

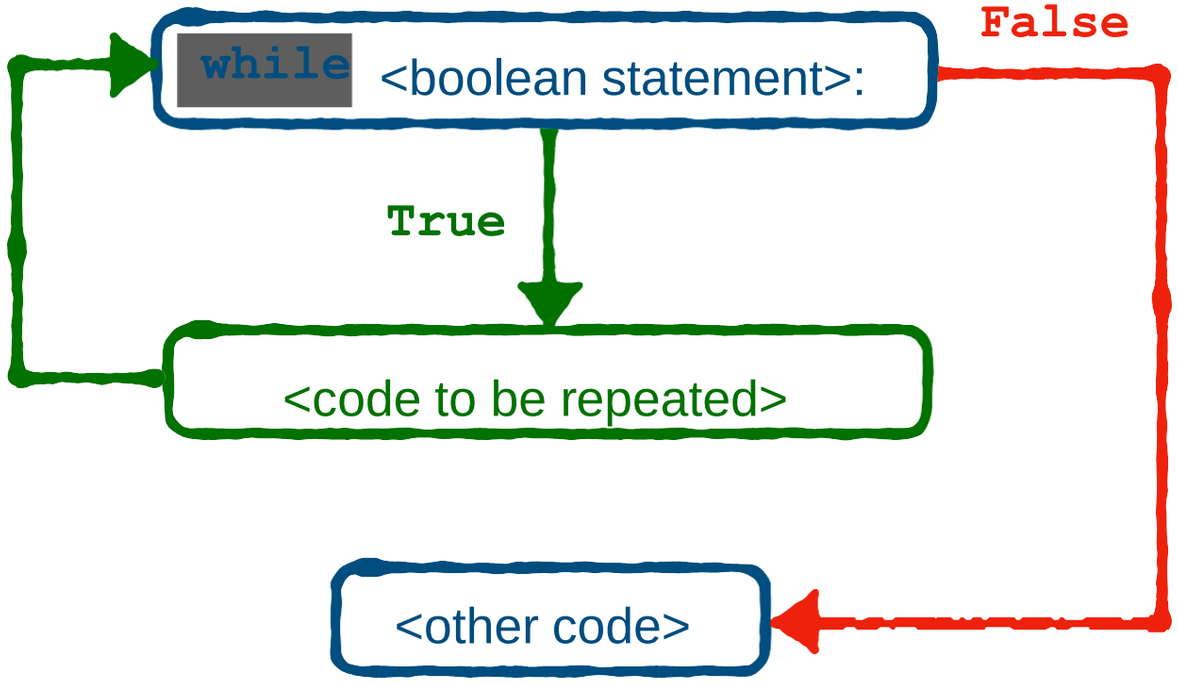
- Iterations and loops
  - The [redacted] loop
    - ✓ Keyword [redacted]

```
[redacted] <boolean statement>:
    <code body to be repeated>
<other code>
```

✓ Check the status of <boolean statement> in each iteration

✓ Repeatedly Execute <code to be repeated> if <boolean statement> is [redacted]

✓ Same indentations for <code to be repeated>



Control Flow: while

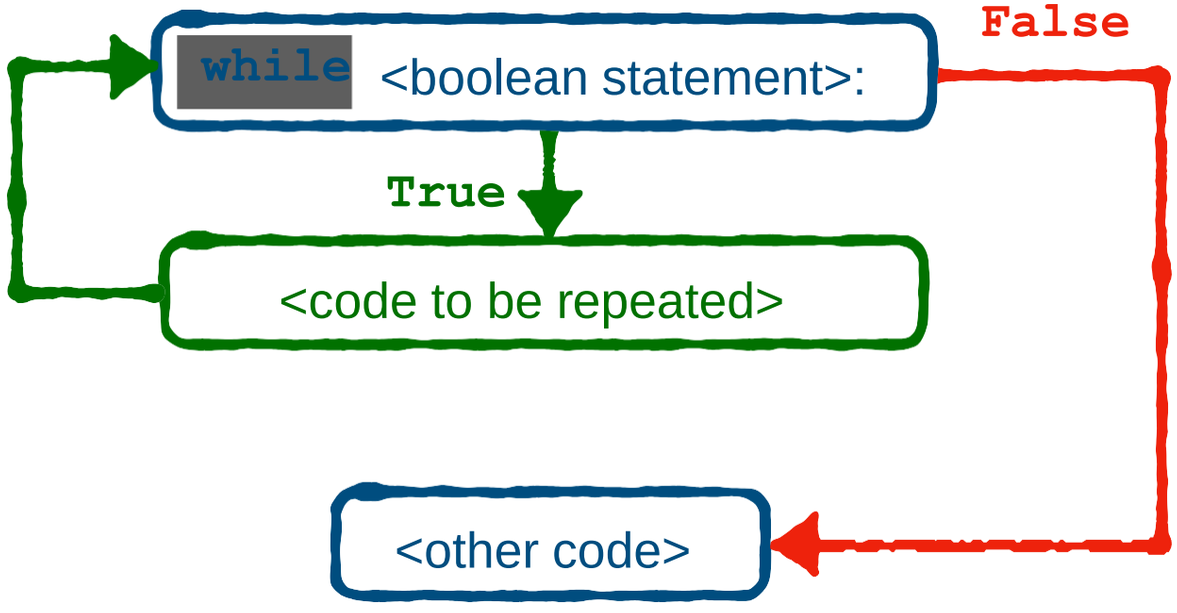
- Iterations and loops
  - Th [redacted] loop
    e

Control Flow: while

- Iterations and loops
  - Th [redacted] loop
    e

Example 1

In the movie Dr. Strange (2016), Steven Strange beat the supervillain "Dormammu" by keeping annoying him until he was extremely bored and frustrated. Write a program to act like Dr. Strange. The "time" loop can only be broken when you type in "I quit" or "You win".

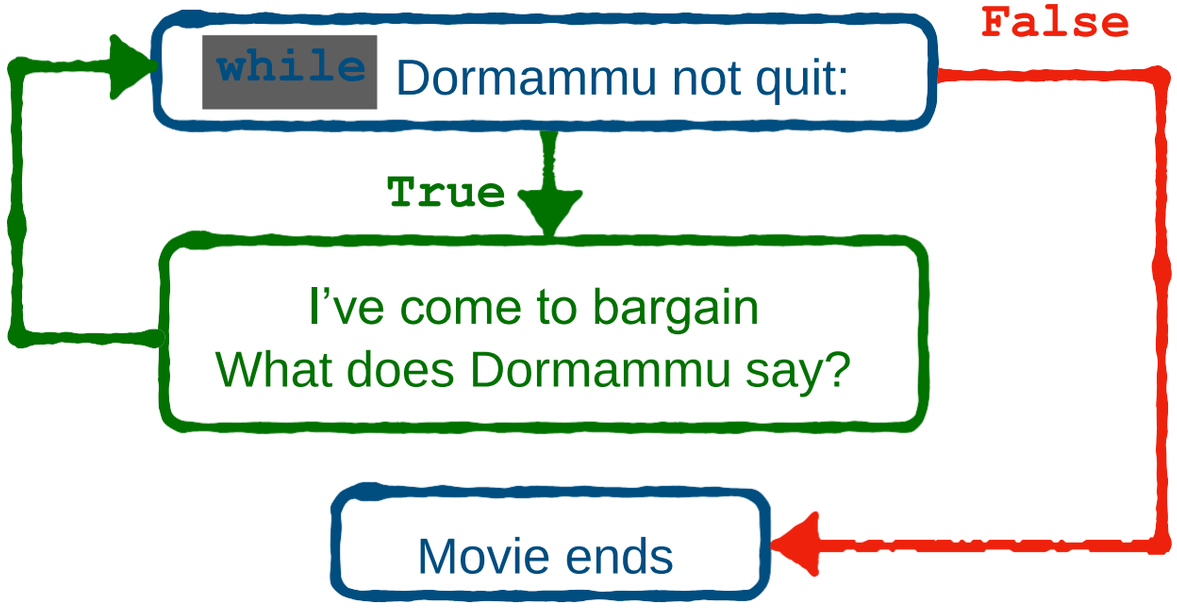


Control Flow: while

- Iterations and loops
  - Th [redacted] loop
    e

Example 1

In the movie Dr. Strange (2016), Steven Strange beat the supervillain "Dormammu" by keeping annoying him until he was extremely bored and frustrated. Write a program to act like Dr. Strange. The "time" loop can only be broken when you type in "I quit" or "You win".



Control Flow: while

- Iterations and loops
  - Th [redacted] loop
    e

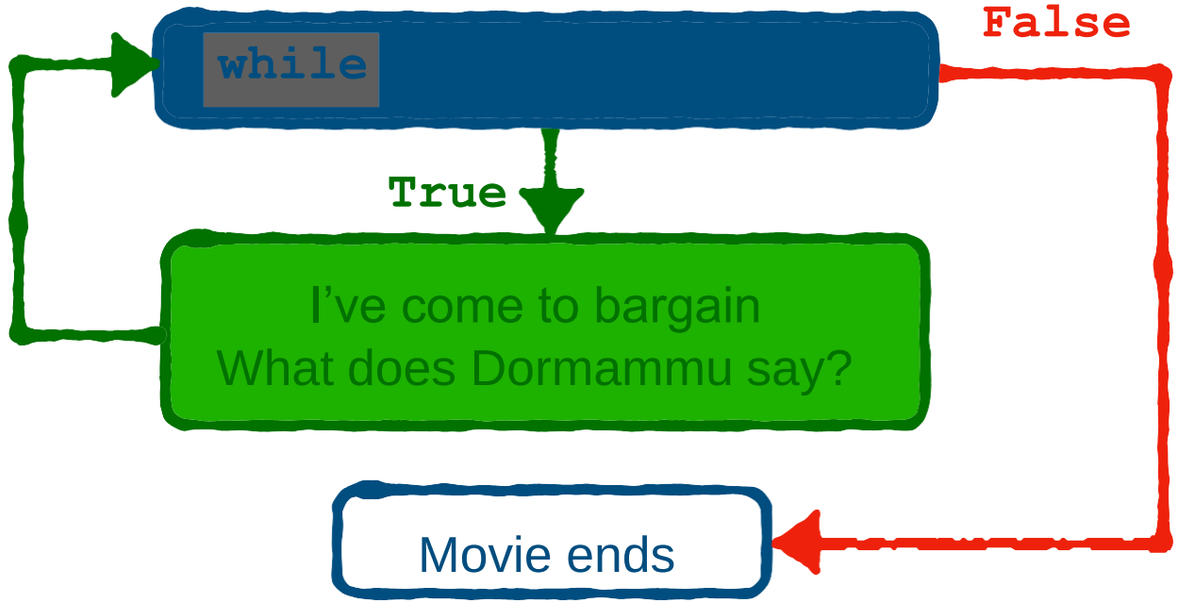
Example 1

In the movie Dr. Strange (2016), Steven Strange beat the supervillain "Dormammu" by keeping annoying him until he was extremely bored and frustrated. Write a program to act like Dr. Strange. The "time" loop can only be broken when you type in "I quit" or "You win".

In [1]:
```
Dormammu_quit = False

while not Dormammu_quit:
    print("Dr. Strange: Dormammu, I've come to bargain!")
    Dormammu_says = input("Dormammu: ")
    if Dormammu_says == 'I quit' or Dormammu_says == 'You win':
        Dormammu_quit = True
        print('Dr. Strange: Wise choice bro!')
    else:
        print("Dr. Strange: Ah~~~~~~~~")
    print('\n')
```

Control Flow: while

- Iterations and loops
  - The [redacted] loop




```
In [1]: Dormammu_quit = False

        while [redacted]
            [redacted]
```

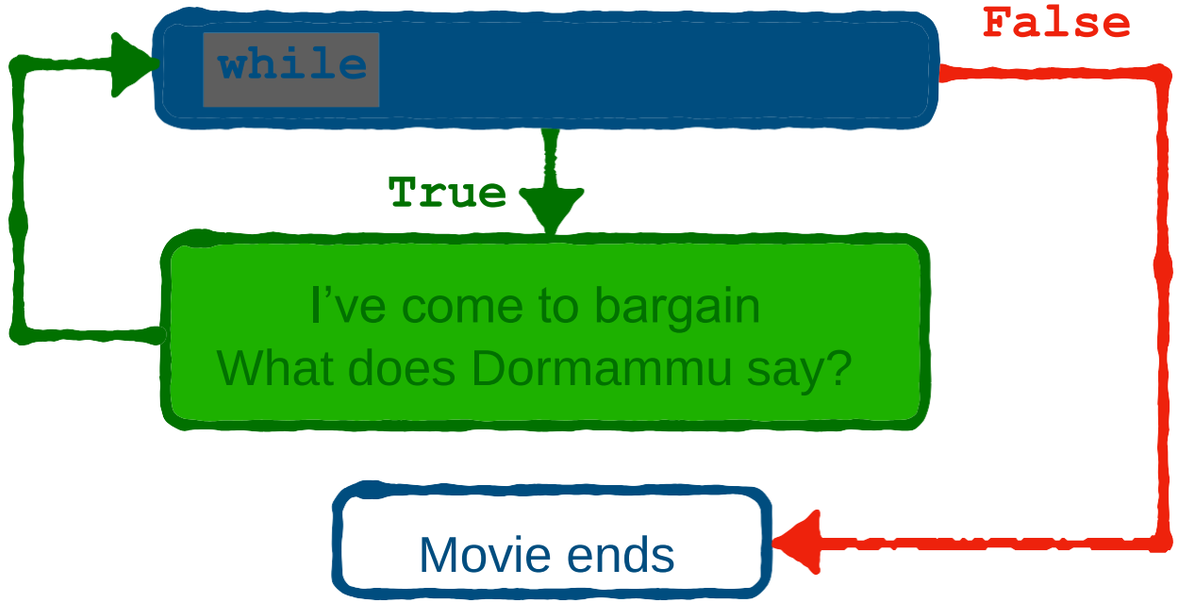
Check the condition in each iteration

Indentation

Loop body

Control Flow: while

- Iterations and loops
  - The [redacted] loop

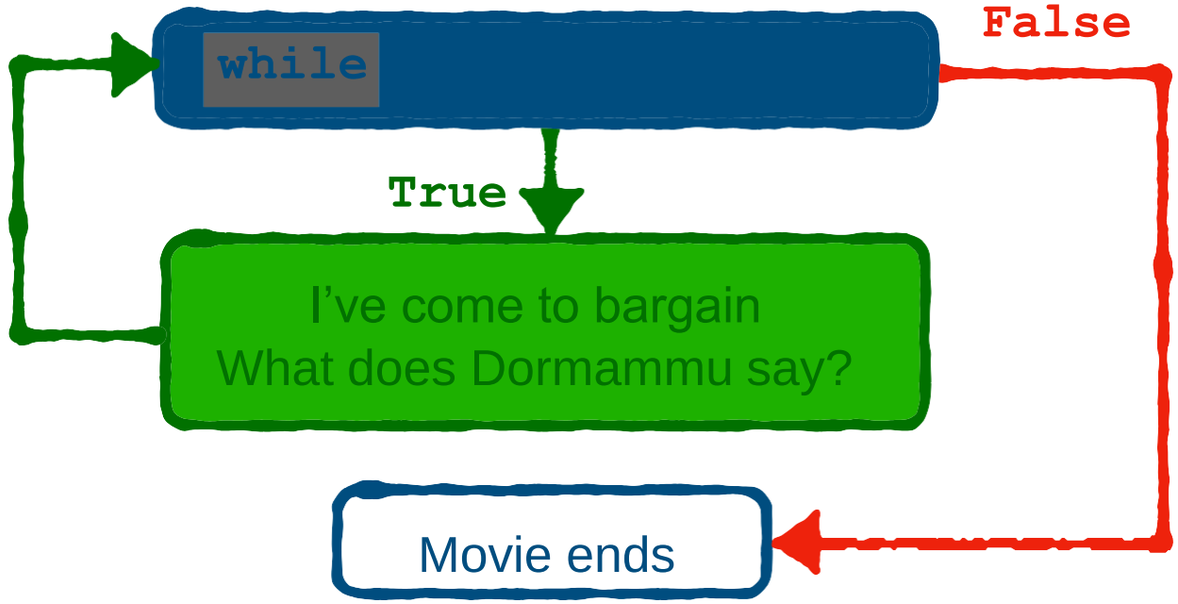


An inner conditional statement to check if Dormammu quits

```
In [1]: Dormammu_quit = False

        while not Dormammu_quit:
            print("Dr. Strange: Dormammu, I've come to bargain!")
            Dormammu_says = input("Dormammu: ")
            [redacted]
            print('\n')
```

Control Flow: while

- Iterations and loops
  - The [redacted] loop




```
In [1]: Dormammu_quit = False

        while not Dormammu_quit:
            print("Dr. Strange: Dormammu, I've come to bargain!")
            Dormammu_says = input("Dormammu: ")
            if Dormammu_says == 'I quit' or Dormammu_says == 'You win':
                [redacted]
            else:
                [redacted]
            print('\n')
```

Extra Indentation

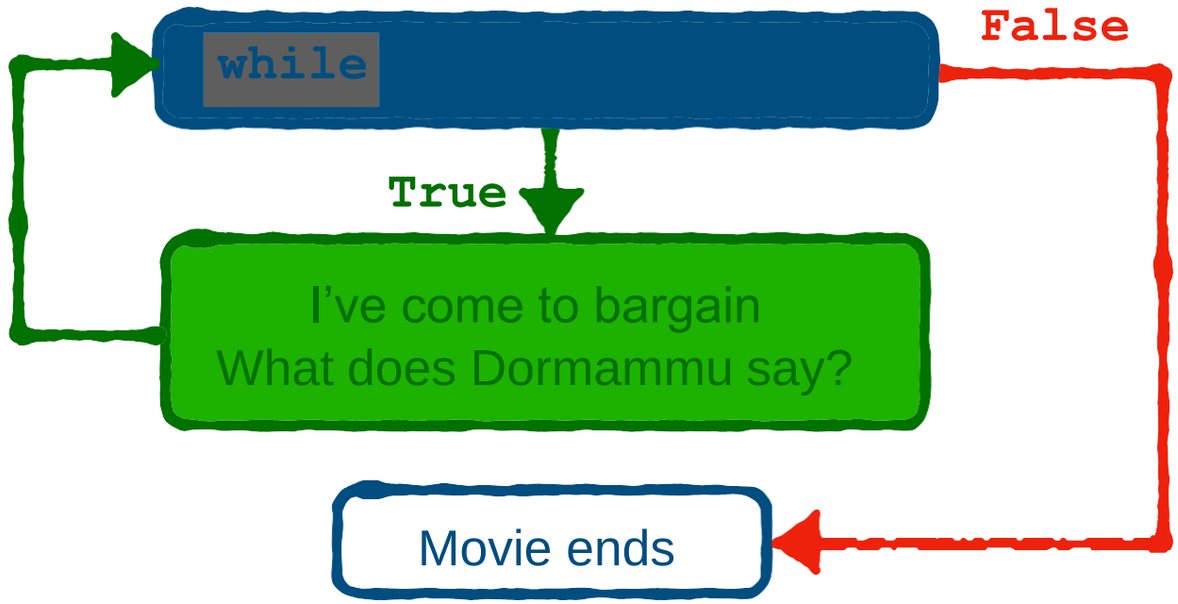
Extra Indentation

Execute if the boolean condition is True

Execute if the boolean condition is False

Control Flow: while

- Iterations and loops
  - ► Th [redacted] loop
    e




```
In [1]: Dormammu_quit = False

        while not [redacted]:
            print("Dr. Strange: Dormammu, I've come to bargain!")
            Dormammu_says = input("Dormammu: ")
            if Dormammu_says == 'I quit' or Dormammu_says == 'You win':
                [redacted]
                print('Dr. Strange: Wise choice bro!')
            else:
                print("Dr. Strange: Ah~~~~~~~~")
            print('\n')
```

True

Change the boolean status to stop the loop

Control Flow: while

- Iterations and loops
  - Th [redacted] loop
    e

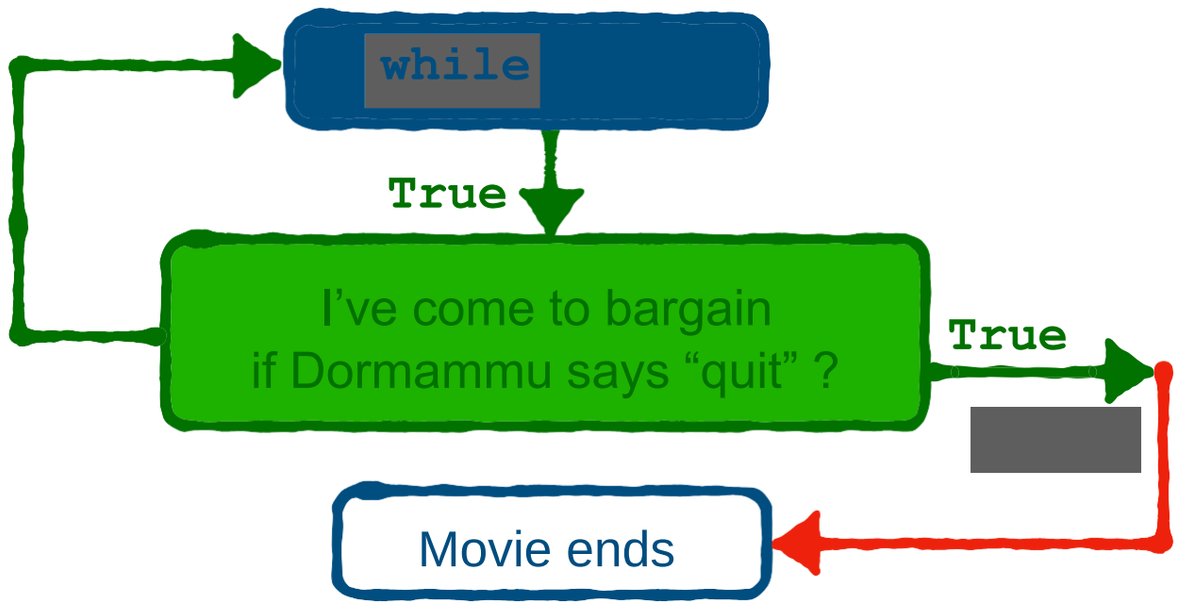
Example 1

In the movie Dr. Strange (2016), Steven Strange beat the supervillain "Dormammu" by keeping annoying him until he was extremely bored and frustrated. Write a program to act like Dr. Strange. The "time" loop can only be broken when you type in "I quit" or "You win".

```
while True:
    print("Dr. Strange: Dormammu, I've come to bargain!")
    Dormammu_says = input("Dormammu: ")
    if Dormammu_says == 'I quit' or Dormammu_says == 'You win':
        print('Dr. Strange: Wise choice bro!')
        break       # It breaks the time loop!
    else:
        print("Dr. Strange: Ah~~~~~~~~")
    print('\n')
```

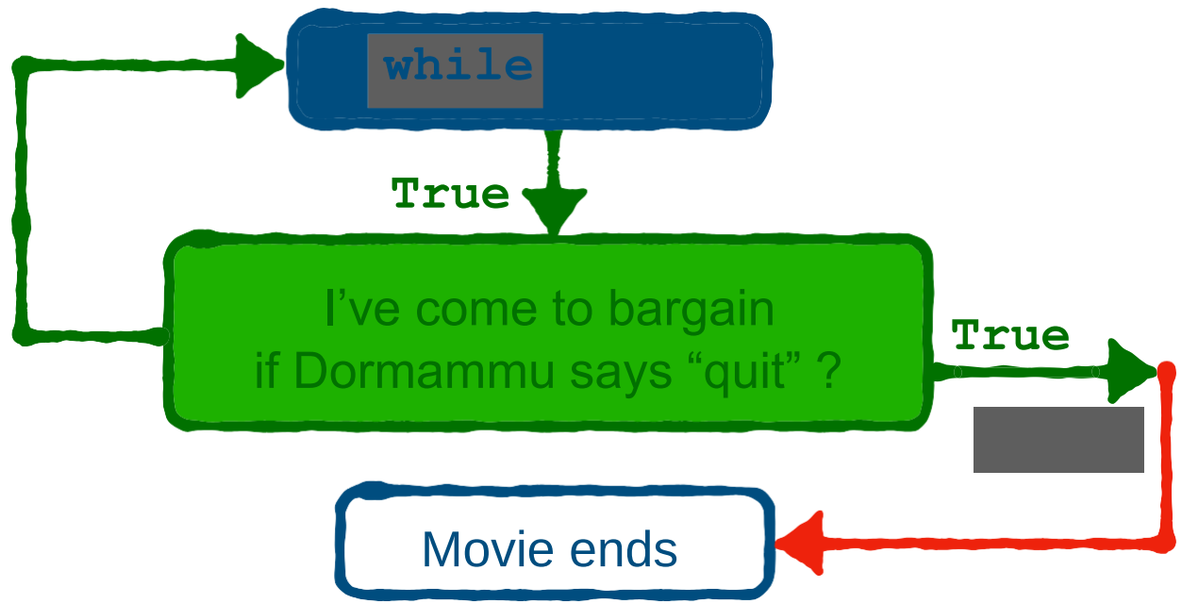
Control Flow: while

- Iterations and loops
  - Th [redacted] loop e



Control Flow: while

- Iterations and loops
  - Th [redacted] loop e



Stop the loop immediately

```
while True:
    print("Dr. Strange: Dormammu, I've come to bargain!")
    Dormammu_says = input("Dormammu: ")
    if Dormammu_says == 'I quit' or Dormammu_says == 'You win':
        print('Dr. Strange: Wise choice bro!')
        [redacted]  # It breaks the time loop!
    else:
        print("Dr. Strange: Ah~~~~~~~~")
    print('\n')
```

Control Flow: while

- Iterations and loops
  - ► Th [redacted] loop
    e
    ✓ Keyword [redacted]
    ✓ Indentations for the code block to be repeated
    ✓ Stoping criterion
    ✓ Indefinite iterations (unknown number of iterations)

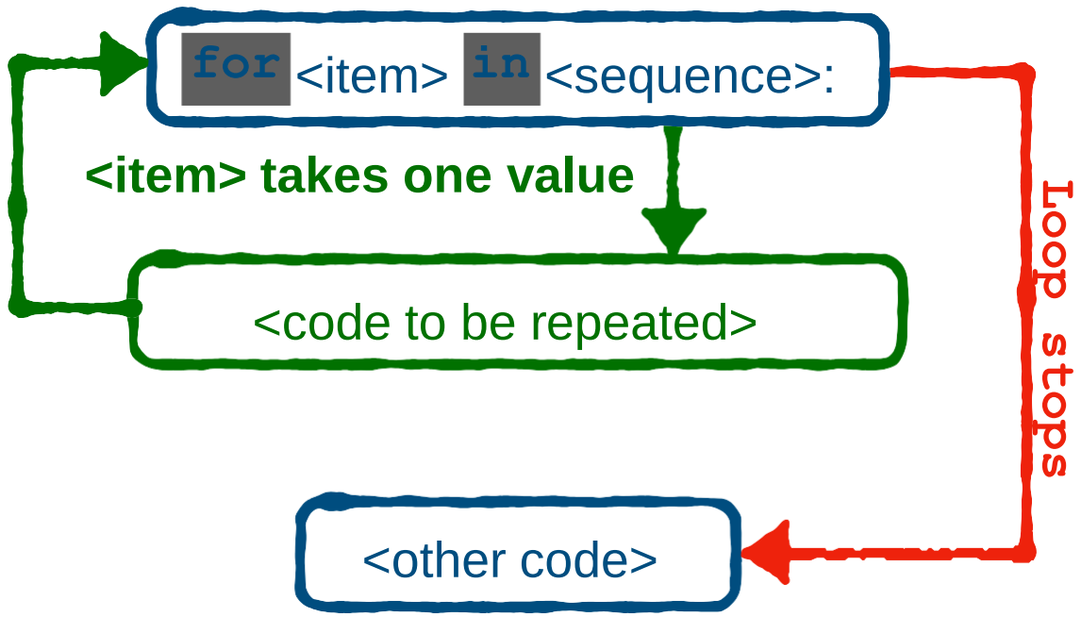
Roadmap

1. Introduction
2. Objects, Variables, and Data Types
3. Strings
4. **Control Flow: for**

Control Flow: for

- Iterations and loops
  - The [for] loop
    - ✓ Keywords [for] and [in]
    - ✓ In each iteration, <item> takes one value from <sequence>
    - ✓ The number of iterations is the number of values in <sequence>
    - ✓ Same indentations for <code to be repeated>

```
[for] <item> [in] <sequence>:
    <code body to be repeated>
<other code>
```

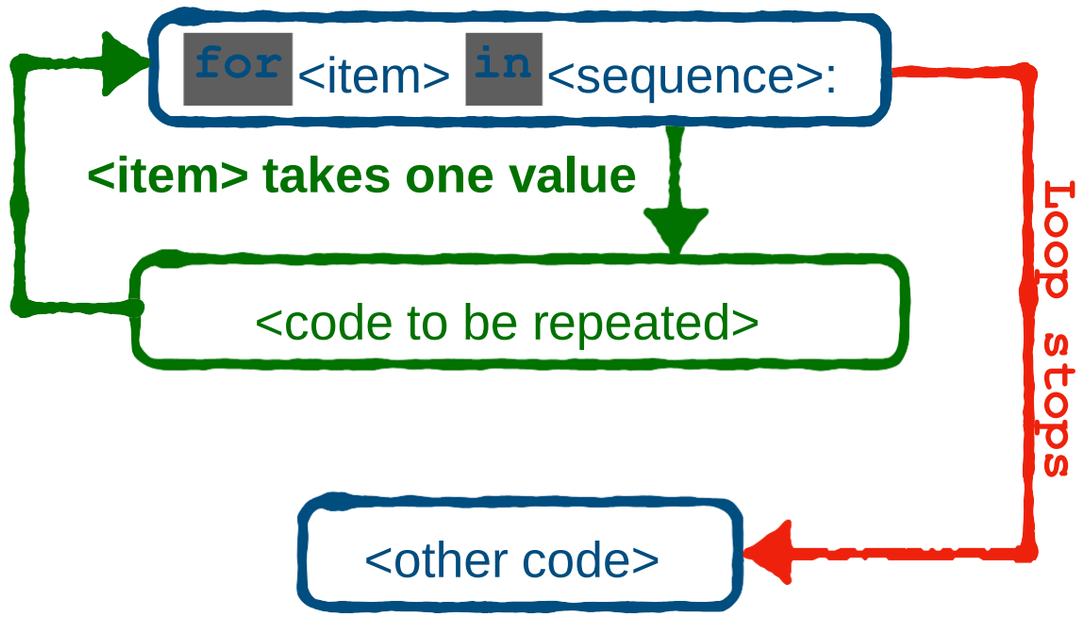


Control Flow: for

- Iterations and loops
  - The `for` loop

Example 2

Cheerleader chant

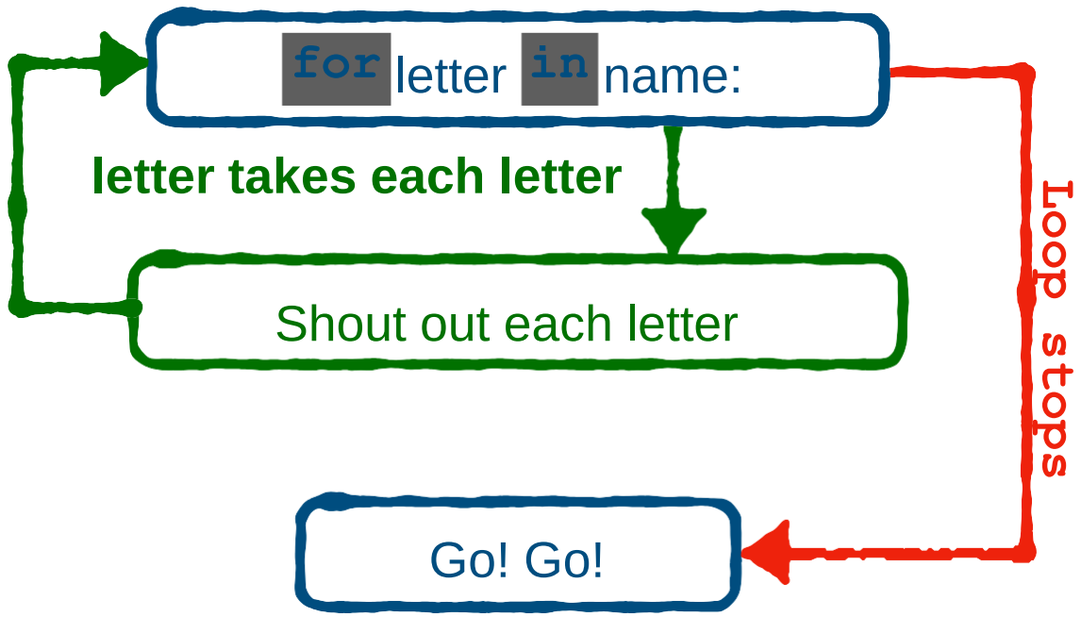


Control Flow: for

- Iterations and loops
  - The `for` loop

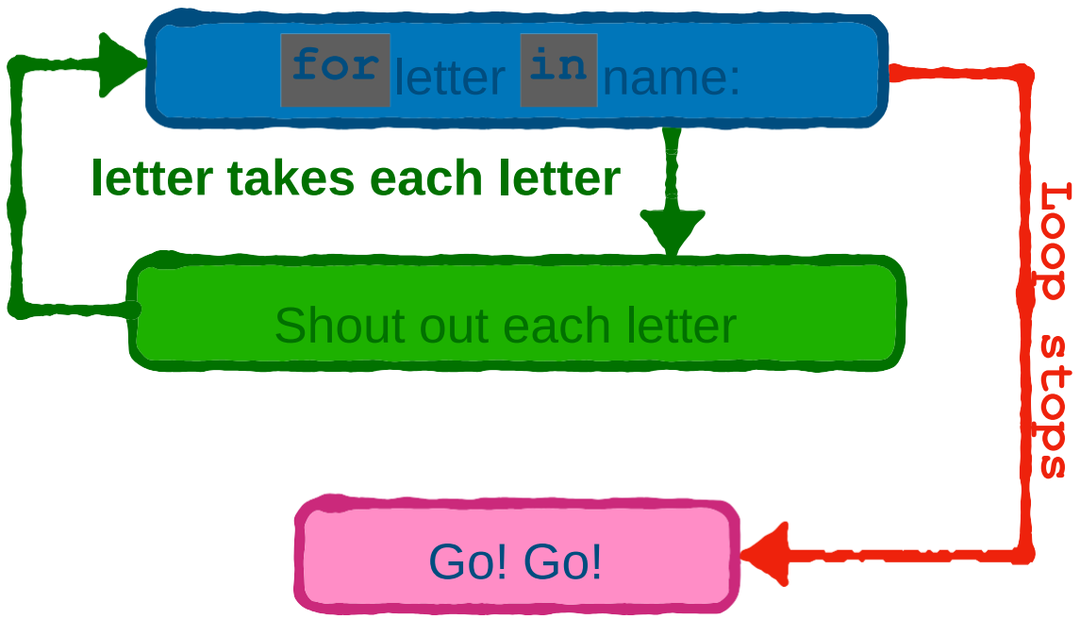
Example 2

Cheerleader chant



Control Flow: for

- Iterations and loops
  - The [redacted] loop



Take each letter from name in each iteration

Indentation

Loop body

```
In [3]: name = input("What is your name? ")

        [redacted]     # Iterate each letter in name
        [redacted]

        print("What's that spell?")
        print(name + "!!")

        print("Go! Go! " + name + "!!")
```

Control Flow: for

- Iterations and loops
  - The [redacted] loop

Take each letter from name in each iteration

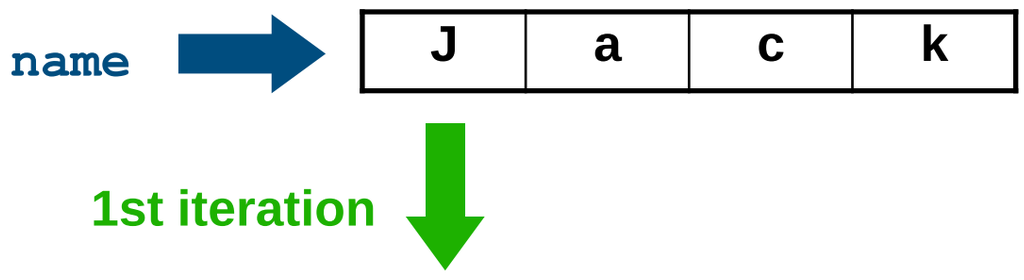
```
In [3]: name = input("What is your name? ")

        [redacted]     # Iterate each letter in name
        [redacted]

        print("What's that spell?")
        print(name + "!!")

        print("Go! Go! " + name + "!!")
```

Indentation

Loop body

Control Flow: for

- Iterations and loops
  - The [redacted] loop



Take each letter from name in each iteration

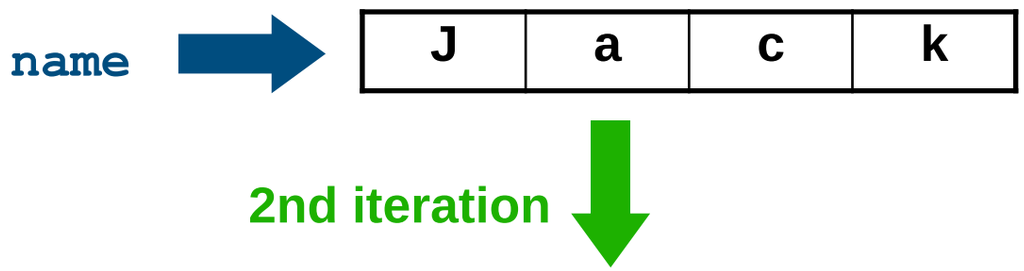
```
In [3]: name = input("What is your name? ")

        [redacted]      # Iterate each letter in name
            [redacted]

        print("What's that spell?")
        print(name + "!!")

        print("Go! Go! " + name + "!!")
```

Indentation

Loop body

Control Flow: for

- Iterations and loops
  - The [redacted] loop



Take each letter from name in each iteration

```
In [3]: name = input("What is your name? ")

        [redacted]      # Iterate each letter in name
        [redacted]

        print("What's that spell?")
        print(name + "!!")

        print("Go! Go! " + name + "!!")
```

Indentation

Loop body

Control Flow: for

- Iterations and loops
  - The [redacted] loop

Take each letter from name in each iteration

```
In [3]: name = input("What is your name? ")

        [redacted]     # Iterate each letter in name
        [redacted]

        print("What's that spell?")
        print(name + "!!")

        print("Go! Go! " + name + "!!")
```

Indentation

Loop body

Control Flow: for

- Iterations and loops
  - The [redacted] loop

Take each letter from name in each iteration

```
In [3]: name = input("What is your name? ")

        [redacted]     # Iterate each letter in name
        [redacted]

        print("What's that spell?")
        print(name + "!!")

        print("Go! Go! " + name + "!!")
```

Indentation

Loop body

Control Flow: for

- Iterations and loops
  - Th [redacted] loop
    e
    ✓ How to skip whitespaces?

Control Flow: for

- Iterations and loops
  - Th [redacted] loop e

Take each letter from name in each iteration

```
In [4]: name = input("What is your name? ")

        # Iterate each letter in name
        for letter in name:
            print(letter)

        print("What's that spell?")
        print(name + "!!")

        print("Go! Go! " + name + "!!")
```

Indentation

Loop body

Control Flow: for

- Iterations and loops
  - Th [redacted] loop
e

In [4]:
```
name = input("What is your name? ")

for letter in name:    # Iterate each letter in name
    if letter == " ":
        continue
    print("Give me a " + letter + ": ")
    print(letter + "!!!")

print("What's that spell?")
print(name + "!!")

print("Go! Go! " + name + "!!")
```

If letter is a whitespace

Extra Indentation

Skip the subsequent code and continue to the next iteration

Control Flow: for

- Iterations and loops
  - [redacted] and [redacted]

Control Flow: for

- Iterations and loops
  - [redacted] and [redacted]

Example 3

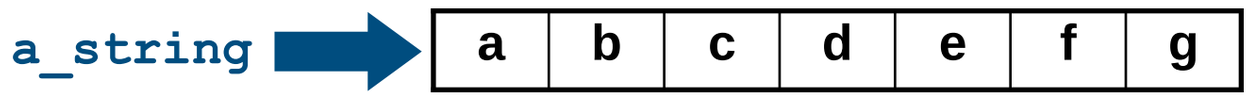
Difference between [redacted] and [redacted]

```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        [redacted]
    new_string = new_string + letter
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        [redacted]
    new_string = new_string + letter
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops



  - [redacted] and [redacted]

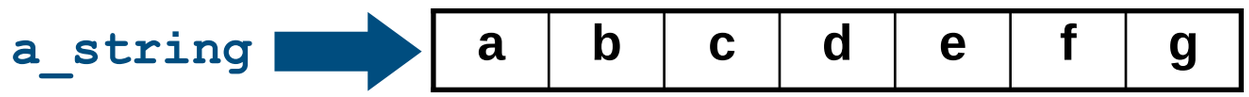
In [5]:

```
new_string = ''
for letter in a_string:
    if letter == 'c':
        break           # Skip the line below and terminate the loop
    new_string = new_string + letter
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        continue        # Skip the line below and go to the next iteration
    new_string = new_string + letter
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops



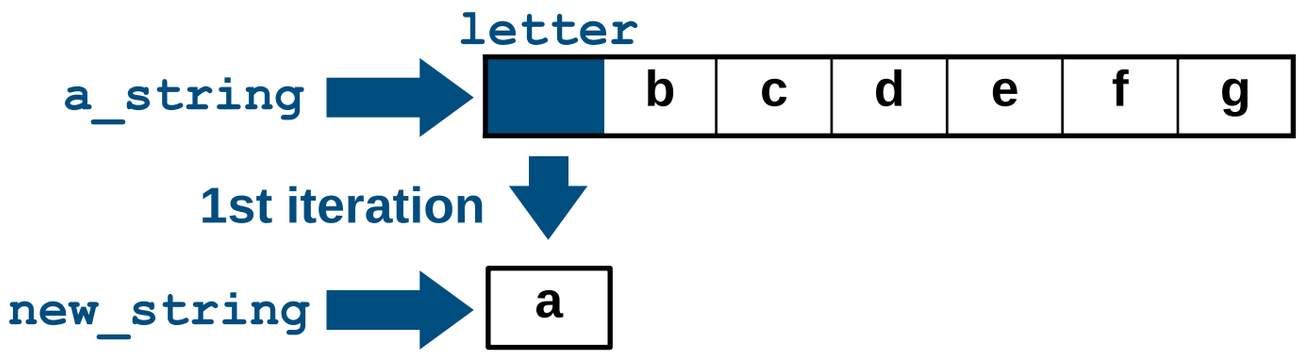
  - [redacted] and [redacted]


```
In [5]: a_string = 'abcdefg'

[redacted]
for letter in a_string:
    if letter == 'c':
        break           # Skip the line below and terminate the loop
    new_string = new_string + letter
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        continue        # Skip the line below and go to the next iteration
    new_string = new_string + letter
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops
  - [redacted] and [redacted]

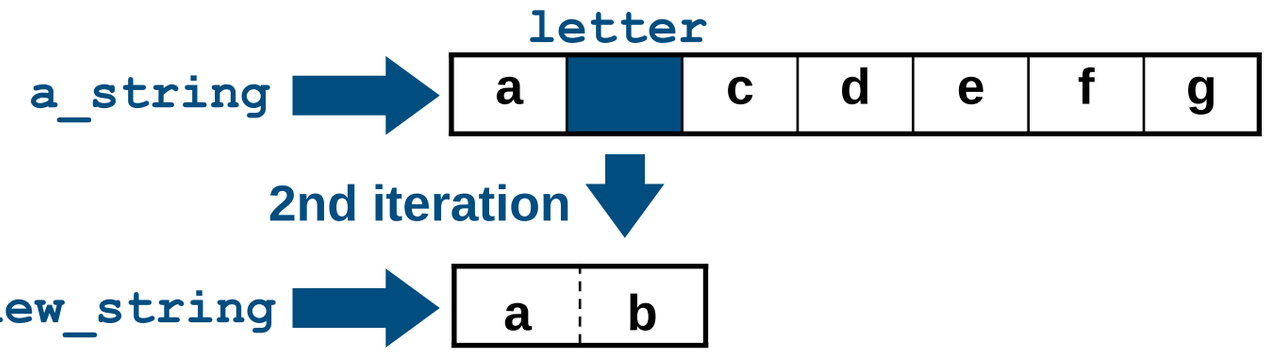



```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        break             # Skip the line below and terminate the loop
    [redacted]
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        continue          # Skip the line below and go to the next iteration
    new_string = new_string + letter
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops
  - [redacted] and [redacted]

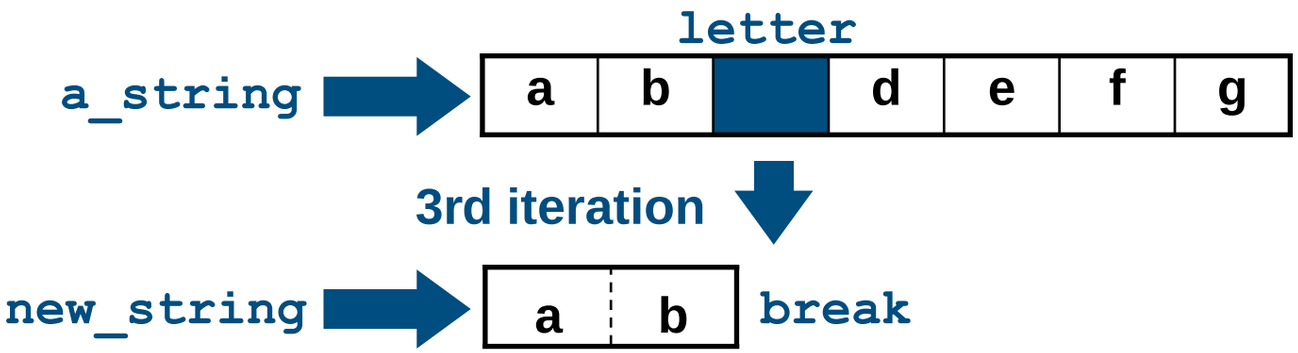



```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        break             # Skip the line below and terminate the loop
    [redacted]
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        continue          # Skip the line below and go to the next iteration
    new_string = new_string + letter
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops
  - [redacted] and [redacted]

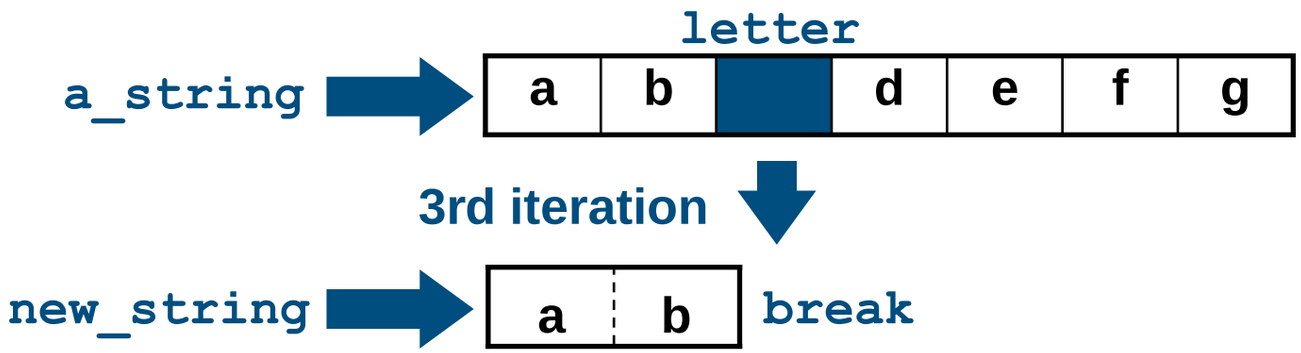



```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        [redacted]              # Skip the line below and terminate the loop
    new_string = new_string + letter
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        continue            # Skip the line below and go to the next iteration
    new_string = new_string + letter
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops
  - [redacted] and [redacted]

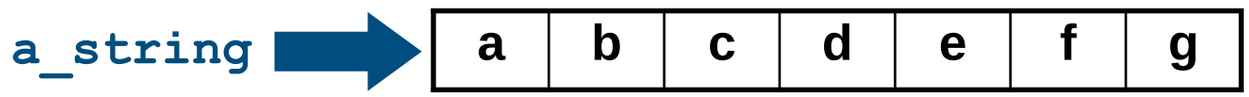



```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        break           # Skip the line below and terminate the loop
    new_string = new_string + letter
[redacted]

new_string = ''
for letter in a_string:
    if letter == 'c':
        continue        # Skip the line below and go to the next iteration
    new_string = new_string + letter
print(new_string)
```

```
[redacted]
abdefg
```

Control Flow: for

- Iterations and loops



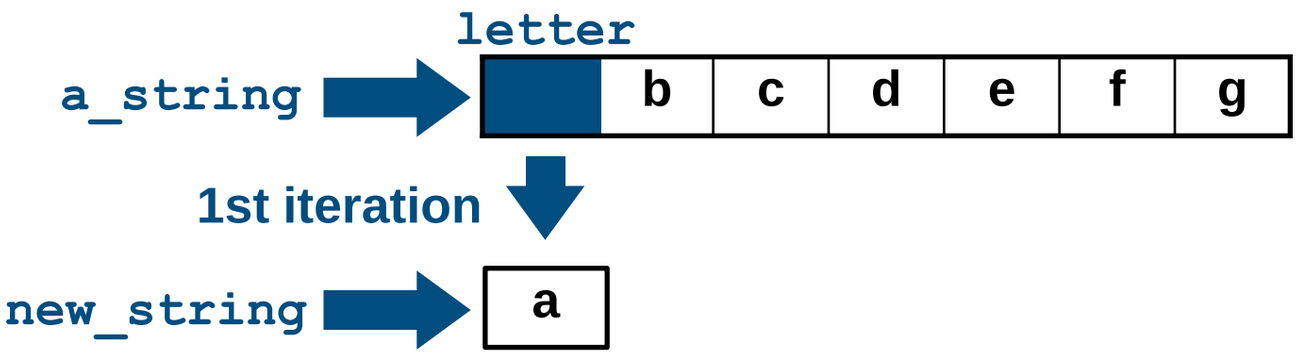
  - [redacted] and [redacted]


```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        break           # Skip the line below and terminate the loop
    new_string = new_string + letter
print(new_string)

[redacted]
for letter in a_string:
    if letter == 'c':
        continue        # Skip the line below and go to the next iteration
    new_string = new_string + letter
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops



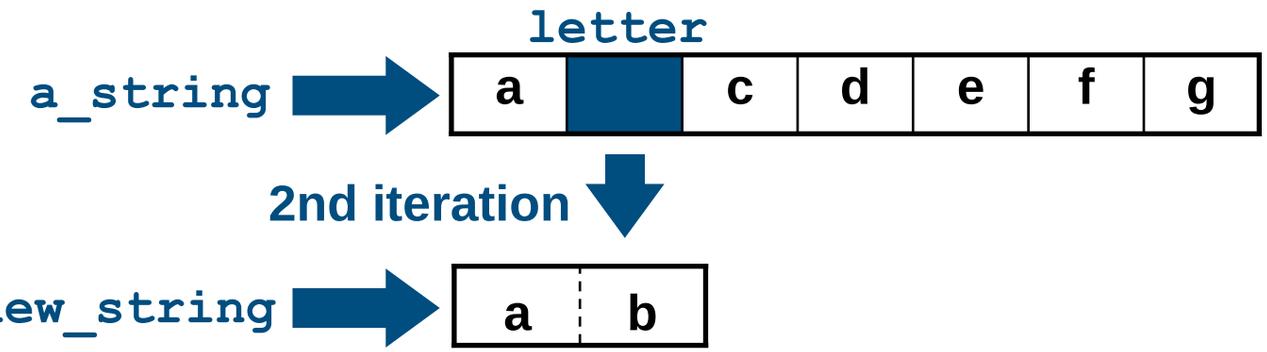
  - [redacted] and [redacted]

```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        break           # Skip the line below and terminate the loop
    new_string = new_string + letter
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        continue        # Skip the line below and go to the next iteration
    [redacted]
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops
  - [redacted] and [redacted]

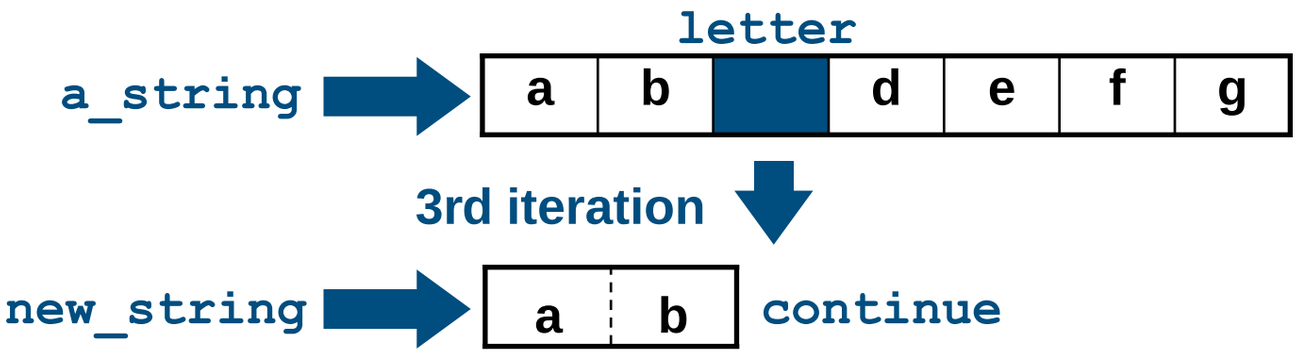



```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        break           # Skip the line below and terminate the loop
    new_string = new_string + letter
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        continue        # Skip the line below and go to the next iteration
    [redacted]
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops
  - [redacted] and [redacted]

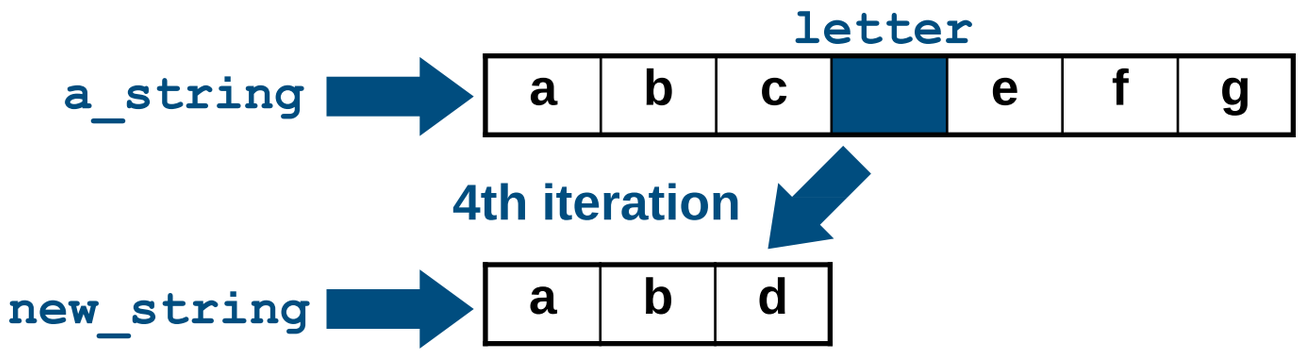



```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        break           # Skip the line below and terminate the loop
    new_string = new_string + letter
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        [redacted]      # Skip the line below and go to the next iteration
    new_string = new_string + letter
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops
  - [redacted] and [redacted]

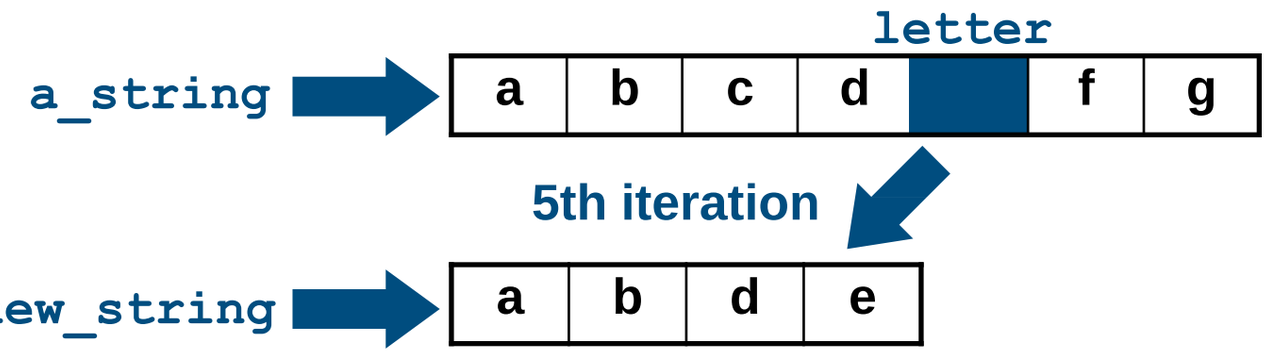



```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        break           # Skip the line below and terminate the loop
    new_string = new_string + letter
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        continue        # Skip the line below and go to the next iteration
    [redacted]
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops
  - [redacted] and [redacted]

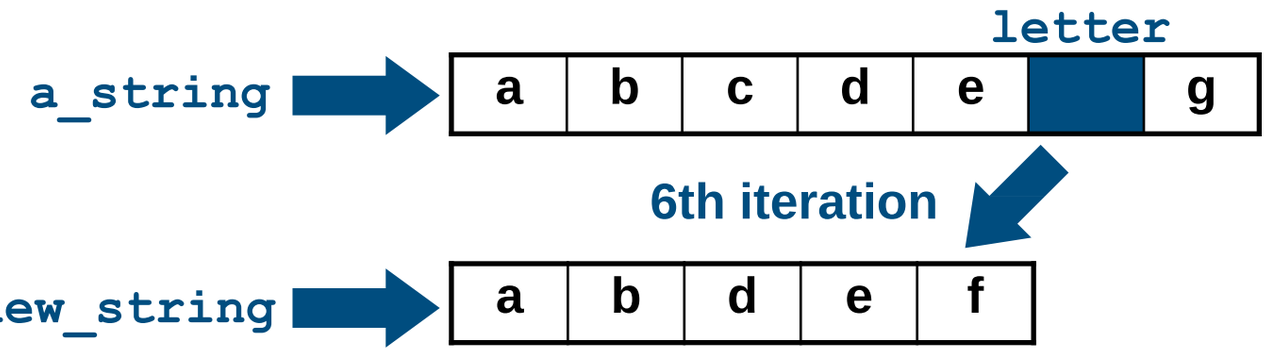



```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        break           # Skip the line below and terminate the loop
    new_string = new_string + letter
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        continue        # Skip the line below and go to the next iteration
    [redacted]
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops
  - [redacted] and [redacted]

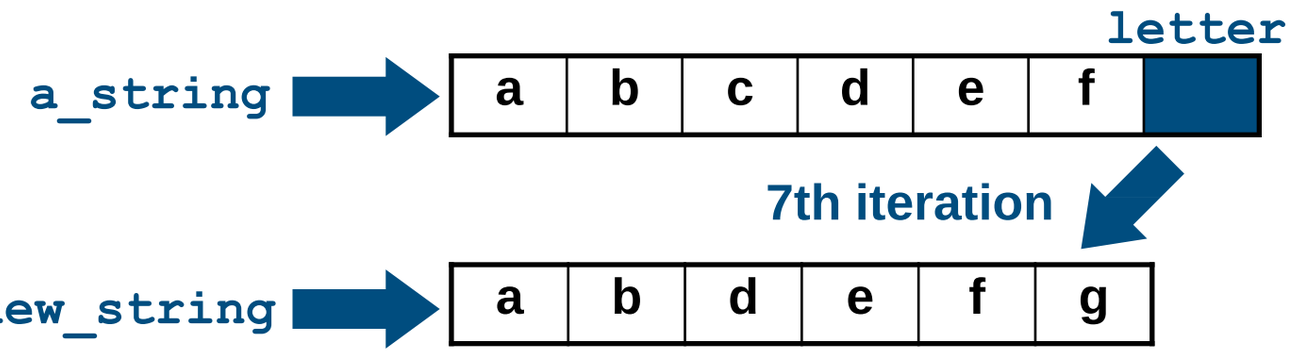



```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        break           # Skip the line below and terminate the loop
    new_string = new_string + letter
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        continue        # Skip the line below and go to the next iteration
    [redacted]
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops
  - [redacted] and [redacted]

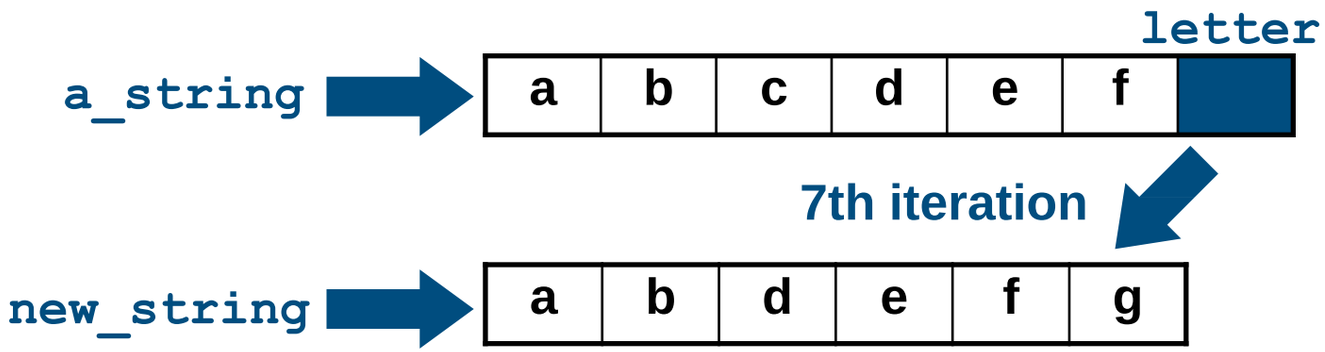



```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        break           # Skip the line below and terminate the loop
    new_string = new_string + letter
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        continue        # Skip the line below and go to the next iteration
    [redacted]
print(new_string)
```

```
ab
abdefg
```

Control Flow: for

- Iterations and loops
  - [redacted] and [redacted]




```
In [5]: a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    if letter == 'c':
        break           # Skip the line below and terminate the loop
    new_string = new_string + letter
print(new_string)

new_string = ''
for letter in a_string:
    if letter == 'c':
        continue        # Skip the line below and go to the next iteration
    new_string = new_string + letter
[redacted]
```

```
ab
[redacted]
```

Control Flow: for

- Iterations and loops

Question

What is the output message of the following code?

```
a_string = 'abcdefg'

new_string = ''
for letter in a_string:
    new_string = new_string + letter
    if letter == 'c':
        break

print(new_string)
```

A. ab

B. abc

C. abcd

D. abdefg

Control Flow: for

- Iterations and loops
  - Comparison between **`while`** loop and **`for`** loop

`while` **loop**	**`for`** **loop**
Indefinite iteration	Definite iteration
A counter can be defined and manually updated	A counter is updated through the loop

Break the loop by **`break`**

Skip the code by **`continue`**