Find, index. A string has any amount of data. It has an ID code. It has a file name extension. It has a keyword. This data must be searched for.

With find, and its friend rfind, we scan strings. If the substring is found, find() returns its index. If no match is located, it returns -1. Its parameters indicate a range.

First example. Here we declare a string that has the word "picture" in it twice. We use find to locate the first occurrence (index 4). And then we find the following occurrence.

Tip:

With the second call to find, please notice the argument "i + 1". This is where the search begins.

Info:

The values returned are 4 and 19. Only the first letter index is returned, not a range.

Note:

If you count the characters in the string (starting at 0), you will see that the P letters are located in those positions.

Based on:

```
Python 3
```

Python program that uses string find

```
value = "cat picture is cat picture"

# Find first index of this string.
i = value.find("picture")
print(i)

# Find first index (of this string) after previous index.
```

```
b = value.find("picture", i + 1)
print(b)
```

Output

```
4
19
```

Not found. Find returns -1 if no value is found. We must check this value, often in an if-statement, to detect when the string was not found. Usually we cannot just use the index directly.

Here:
The string "python" is not found within the string. The value of the variable i is thus equal to negative one.

Python program that tests find result

```
value = "ralph waldo emerson"
i = value.find("python")

if i != -1:
    # Not reached.
    print("String found")
else:
    print("String not found")
```

Output

```
String not found
```

While. Suppose we want to loop over all instances of a string within another string. A while-loop with find can accomplish this. We use the result of find to advance the starting index.

While

Tip:
We could optimize this sample further. Try changing the second argument to find to add the length of string.

And:
This will avoid searching all the characters within a found substring. This avoids finding overlapping strings.

Python program that uses string find, while

```
value = "cat picture is cat picture"

# Start with this value.
location = -1
```

```
# Loop while true.
while True:
    # Advance location by 1.
    location = value.find("picture", location + 1)

    # Break if not found.
    if location == -1: break

    # Display result.
    print(location)
```

Output

```
4
19
```

Rfind. This method searches from the right. It returns the index of the rightmost substring within the (optional) range specified.

Please note:

The integer arguments are a range. We specify the first index and the last index.

Here:

In this example, we call rfind twice. In the second call, we specify two range arguments.

Tip:

This is like a slice we search. We stop searching one index before the location of the first instance.

Python program that uses string rfind

```
value = "cat picture is cat picture"

# Get rightmost index of this string.
i = value.rfind("picture")
print(i)

# Get rightmost index within this range of characters.
# ... We search the left four words.
b = value.rfind("picture", 0, i - 1)
print(b)
```

Output

```
19
4
```

Rfind, loop. We can use the rfind method in a loop. Here we adjust the range of characters we search as we progress through the string.

And:

We adjust the end index, but leave the first index set to 0. Thus

we iterate over matched substrings from the right.

Python program that uses rfind, while

```
value = "cat picture is cat picture"

# Start with length of string.
i = len(value)

while True:
    # Find rightmost string in this range.
    i = value.rfind("picture", 0, i)

    # Check for not found.
    if i == -1: break
    print(i)
```

Output

```
19
4
```

Index. This method is the same as find on strings, except for one big difference. Index() raises an error when the string is not found.

Note:

In most programs, checking for negative one is better. Avoiding exceptions improves performance.

Rindex:

As with find and rfind, there is an rindex method available. This searches from the right, not the left.

Python program that uses string index

```
value = "abc def"

# Use index method.
i = value.index("def")
print(i)

# This causes an exception.
b = value.index("xyz")
```

Output

```
4
Traceback (most recent call last):
  File "C:\programs\file.py", line 11, in <module>
    b = value.index("xyz")
ValueError: substring not found
```

In-operator. This can also search strings. It returns no index. It simply returns True if the string is found in the source string, and False if not.

In

Note:

The in-operator has simpler syntax. It is often preferred if no index is required.

Here:

We use "in" and "not in" to see if a string contains certain file extensions (these may be anywhere in the string).

Python that uses in and not in on strings

```
filename = "cat.png"

# See if the string contains this substring.
if ".png" in filename:
    print("Is PNG image")

# This is evaluated to true.
if ".jpg" not in filename:
    print("Is NOT JPG image")
```

Output

```
Is PNG image
Is NOT JPG image
```

Searching. A string can be searched in many ways. With find, and its friend rfind, we get the index of a located match. With "in," we see if the string exists.

With index and rindex, we get an error when no match is located. Find returns negative one in that situation. We used these methods within loops and iterated with them.