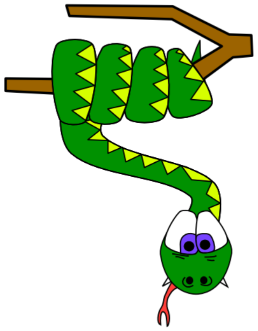


String

Session-3

Strings

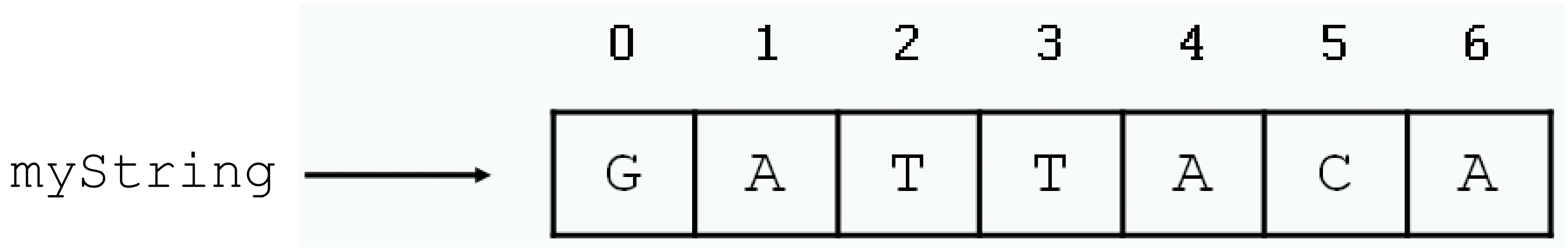
- A <u>string</u> is a sequence of letters (called <u>characters</u>).
- In Python, strings start and end with single or double quotes.

```
>>> "foo"
'foo'
>>> 'foo'
'foo'
```

Defining strings

- Each string is stored in the computer's memory as a list of characters.

```
>>> myString = "GATTACA"
```



Accessing single characters

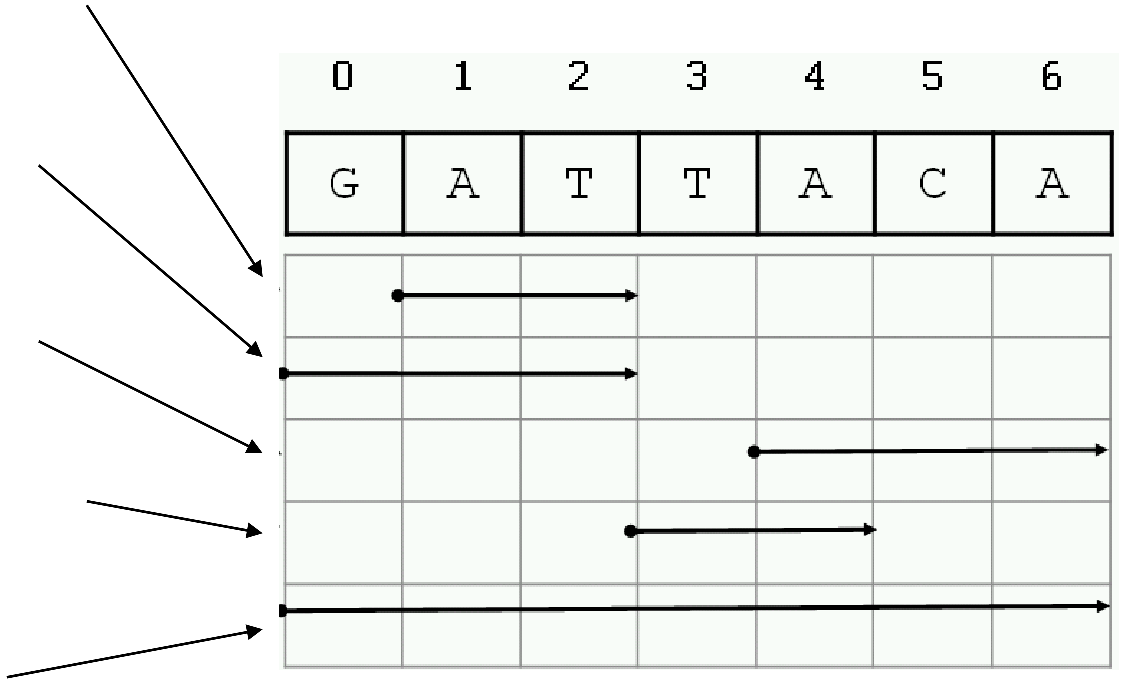
- You can access individual characters by using indices in square brackets.

```
>>> myString = "GATTACA"
>>> myString[0]
'G'
>>> myString[1]
'A'
>>> myString[-1]
'A'
>>> myString[-2]
'C'
>>> myString[7]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: string index out of range
```

Negative indices start at the end of the string and move left.

Accessing substrings

```
>>> myString = "GATTACA"
>>> myString[1:3]
'AT'
>>> myString[:3]
'GAT'
>>> myString[4:]
'ACA'
>>> myString[3:5]
'TA'
>>> myString[:]
'GATTACA'
```



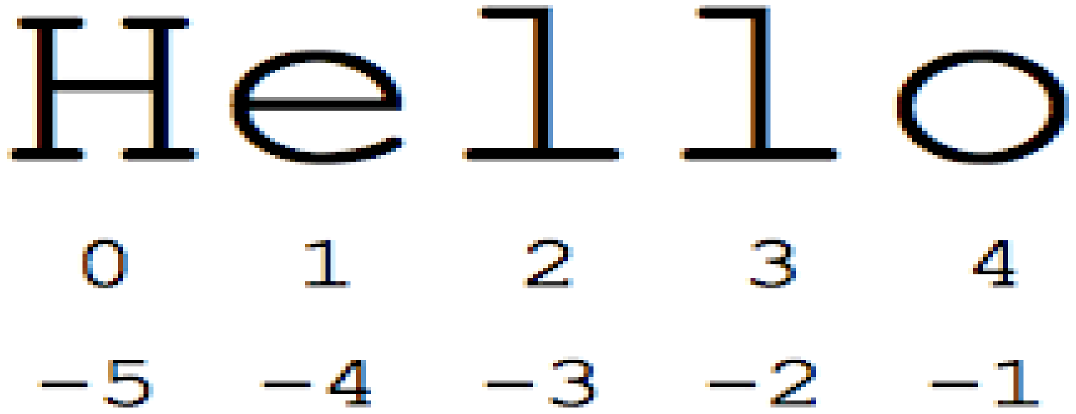
`s[i:j:k]` extracts every `k`th element starting with index `i` (inlcusive) and ending with index `j` (not inclusive)

```
>>> s[0:5:2]
```

Negative Indexing

Python also supports negative indexes. For example, `s[-1]` means extract the first element of `s` from the end (same as `s[len(s)-1]`)

```
>>> s[-1]
'g'
>>> s[-2]
'n'
```



Special characters

- The backslash is used to introduce a special character.

```
>>> "He said, "Wow!""
  File "<stdin>", line 1
    "He said, "Wow!""
                  ^
SyntaxError: invalid
  syntax
>>> "He said, 'Wow!'"
"He said, 'Wow!'"
>>> "He said, \"Wow!\""
'He said, "Wow!"'
```

Escape sequence	Meaning
\\	Backslash
\’	Single quote
\”	Double quote
\n	Newline
\t	Tab

More string functionality

```
>>> len(“GATTACA”)
7
>>> “GAT” + “TACA”
‘GATTACA’
>>> “A” * 10
‘AAAAAAAAAA
>>> “GAT” in “GATTACA”
True
>>> “AGT” in “GATTACA”
False
```

← Length

← Concatenation

← Repeat

← Substring test

String Operations-String Module

- import string
- *#returning all letters*
- print(string.ascii_letters)
- *#returning lowercase letters*
- print(string.ascii_lowercase)
- *#returning uppercase letters*
- print(string.ascii_uppercase)
- *#returning all punctuations*
- print(string.punctuation)
- *#returning whitespaces*
- print(string.whitespace)
- *#returning all digits*
- print(string.digits)

Try This

- *'''checking for whitespaces'''*
- import string
- if " " *in string.whitespace:*
-     print(True)
- for i in string.whitespace:
-     print(repr(i))
- for i in string.punctuation:
-     print(i)
- *"""repr convert special*
- *character into normal"""*

String Operation

```
st="hello world"
st=st.capitalize()#'''capitalizes only first letter'''
print(st)
st=st.title()
print(st)#'''capitaizes all words'''
st=st.lower()#'''covert string to lower case'''
print(st)
st=st.upper()#'''convert in uppercase'''
print(st)
if st.isupper():
    print(True)
st="hello"
x=st.islower()
print(x)
if st.islower():#'''checks whether all characters are lower
case'''
    print("True")
```

More .....

```
st="abc~123"
if st.isalpha():
    print("true")
st="Hello World"
x=st.istitle()#'''check whether string is a title'''
print(x)
st="4564"
x=st.isdigit()
print(x)
```

More…

- S=“hello world”
- S.find(‘h’) –returns index of h, if not found returns -1
- S.index(‘h’)—returns index of h, if not found returns error
- S.rfind(‘o’)—returns rightmost index of the substring
- S.count(‘substring’,start,end)—count nu. Of occurrences of substring between start and end. Ex: S.count(‘l’,6,10) returns 1
- S.count(‘l’) returns 3 [default search in all string]

split()

- The split() method with a string argument separates strings based on the specified delimiter.
- **Note 2:**With no arguments, split() separates strings using one or more spaces as the delimiter.
- Return a list

- s = "topeka,kansas city,wichita,olathe"
-
- # Separate on comma.
- cities = s.**split**(",")
-
- # Loop and print each city name.
- for city in cities:
-     print(city)
-

```
s = "One two    three"

# Call split with no arguments.
words = s.split()

# Display results.
for word in words:
    print(word)
```

rsplit()

- **Rsplit.** Usually rsplit() is the same as split. The only difference occurs when the second argument is specified. This limits the number of times a string is separated.
- **So:**When we specify 3, we split off only three times from the right. This is the maximum number of splits that occur. # Data.
- s = "Buffalo;Rochester;Yonkers;Syracuse;Albany;Schenectady"

- # Separate on semicolon.
- # ... Split from the right, only split three.
- cities = s.rsplit(";", 3)

- # Loop and print.
- for city in cities:
-     print(city)

Splitlines()

- **Splitlines.** Lines of text can be separated with Windows, or UNIX, newline sequences. This makes splitting on lines complex. The splitlines() method helps here.
- # Data.
- s = """This string
- has many
- lines."""
- # Split on line breaks.
- lines = s.splitlines()
- # Loop and display each line.
- for line in lines:
-     print("[" + line + "]")
- **Output**
- [This string]
- [has many]
- [lines.]

Join

- **Join.** This method combines strings in a list or other iterable collection.
- **With join,** we reverse the split() operation. We can use a delimiter of zero, one or more characters.
- list = ["a", "b", "c"]
- # Join with empty string literal.
- result = "".join(list)
- # Join with comma.
- result2 = ",".join(list)
- # Display results.
- print(result)
- print(result2)
- **Output**
- abc
- a,b,c

strip()

- **With strip,** we remove certain characters (such as whitespace) from the left and right parts of strings. We invoke lstrip, rstrip and the strip().
- **Lstrip:**With no argument, lstrip removes whitespace at the start of the string. The L stands for left.
- **Rstrip:**With no argument, rstrip removes whitespace at the end. This is the right side. If no whitespace is present, nothing happens.

Example

- # Has two leading spaces and a trailing one.
- value = " a line "
- # Remove left spaces.
- value1 = value.lstrip()
- print("[" + value1 + "]")
- # Remove right spaces.
- value2 = value.rstrip()
- print("[" + value2 + "]")
- # Remove left and right spaces.
- value3 = value.strip()
- print("[" + value3 + "]")
- **Output**
- [a line]
- [a line]
- [a line]

Example

- # Has numbers on left and right, and some syntax.
- value = "50342=Data,231"

- # Strip all digits.
- # ... Also remove equals sign and comma.
- result = value.strip("0123456789=,")
- print(result)

rjust and ljust

- **Ljust and rjust** pad strings. They accept one or two arguments. The first argument is the total length of the result string. The second is the padding character.
- s = "Paris"

- # Justify to left, add periods.
- print(s.ljust(10, "."))
- # Justify to right.
- print(s.rjust(10))
- #justify center
- print(*"hello".center(10,"."))*
- Output
- Paris.....
-      Paris
- ..hello...

startswith

- phrase = "cat, dog and bird"
- # See if the phrase starts with these strings.
- if phrase.startswith("cat"):
-     print(True)
- if phrase.startswith("cat, dog"):
-     print(True)
- # It does not start with this string.
- if not phrase.startswith("elephant"):
-     print(False)
- **Output**
- True
- True
- False

endswith

- url = "https://www.rediffmail.com/"

- # Test the end of the url.
- if url.endswith("/"):
-     print("Ends with slash")
- if url.endswith(".com/"):
-     print("Ends with .com/")
- if url.endswith("?") == False:
-     # Does not end in a question mark.
-     print(False)
- **Output**

- Ends with slash
- Ends with .com/
- False