Chapter 8 More on Strings and Special Methods

Objectives

- To learn how to create strings (§8.2.1).
- To use the **len**, **min**, and **max** functions to obtain the length of a string or the smallest or largest element in a string (§8.2.2).
- To access string elements by using the index operator (**[]**)(§8.2.3).
- To get a substring from a larger string by using the slicing **str[start:end]** operator (§8.2.4).
- To concatenate strings by using the + operator and to duplicate strings by using the * operator (§8.2.5).
- To use the **in** and **not in** operators to determine whether a string is contained within another string (§8.2.6).
- To compare strings by using comparison operators (==, !=, <, <=, >, and >=) (§8.2.7).
- To iterate characters in a string by using a foreach loop (§8.2.8).
- To test strings by using the methods **isalnum**, **isalpha**, **isdigit**, **isidentifier**, **islower**, **isupper**, and **isspace** (§8.2.9).
- To search for substrings by using the methods **endswith**, **startswith**, **find**, **rfind**, and **count** (§8.2.10).
- To convert strings by using the methods **capitalize**, **lower**, **upper**, **title**, **swapcase**, and **replace** (§8.2.11).

The str Class

Creating Strings

```
s1 = str() # Create an empty string
s2 = str("Welcome") # Create a string Welcome
```

Python provides a simple syntax for creating string using a string literal. For example,

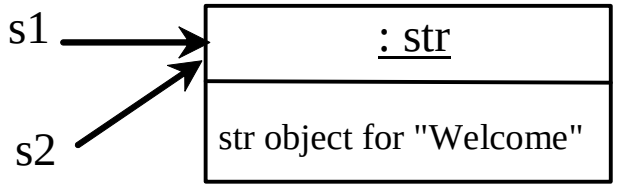
```
s1 = "" # Same as s1 = str()
s2 = "Welcome" # Same as s2 = str("Welcome")
```

Strings are Immutable

A string object is immutable. Once it is created, its contents cannot be changed. To optimize performance, Python uses one object for strings with the same contents. As shown in Figure 6.8, both s1 and s2 refer to the same string object.

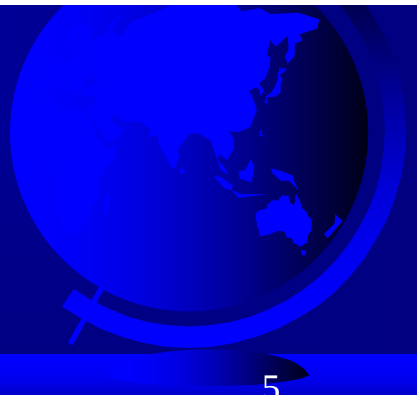
```
>>> s1 = "Welcome"
>>> s2 = "Welcome"
>>> id(s1)
505408902
>>> id(s2)
505408902
```

After executing `s = "HTML";`

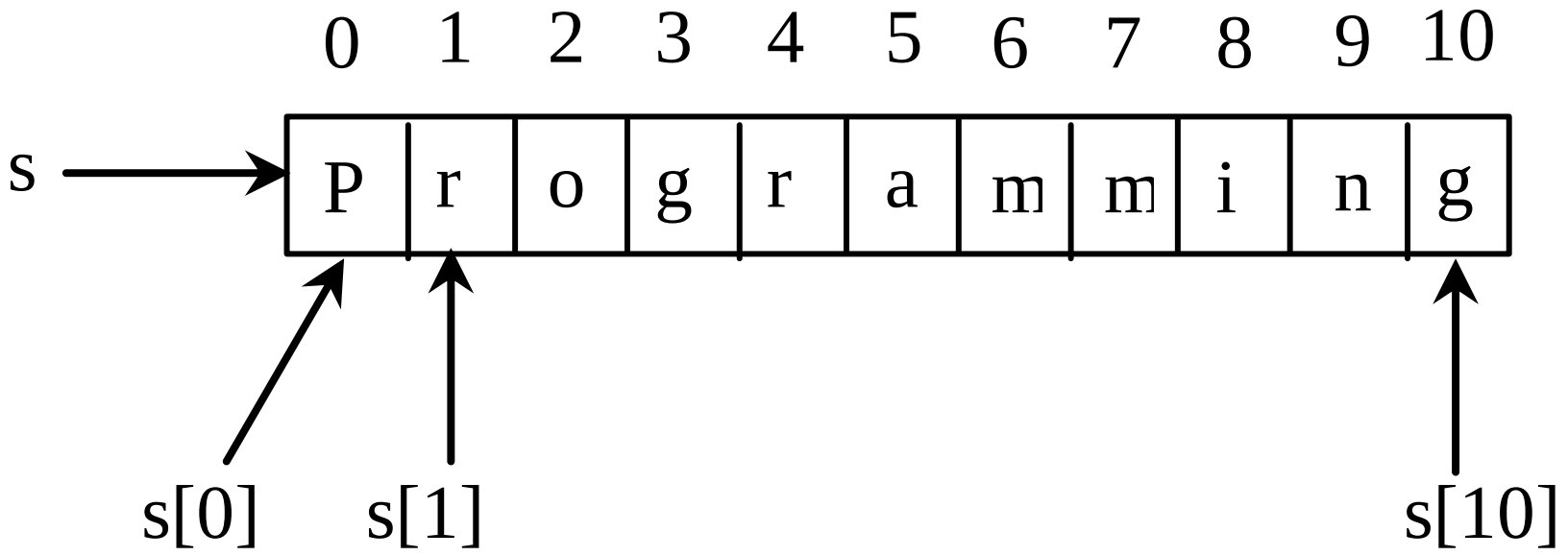


Functions for str

```
>>> s = "Welcome"
>>> len(s)
7
>>> max(s)
o
>>> min(s)
W
```



Index Operator []



The +, *, [:], and in Operators

```
>>> s1 = "Welcome"
>>> s2 = "Python"
>>> s3 = s1 + " to " + s2
>>> s3
'Welcome to Python'
>>> s4 = 2 * s1
>>> s4
'WelcomeWelcome'
>>> s1[3 : 6]
'com'
>>> 'W' in s1
True
>>> 'X' in s1
False
```

Negative Index

```
>>> s1 = "Welcome"
>>> s1[-1]
'e'
>>> s1[-3 : -1]
'me'
```

The in and not in Operators

```
>>> s1 = "Welcome"
>>> "come" in s1
True
>>> "come" not in s1
False
>>>
```

Foreach Loops

```
for ch in string:
    print(ch)
```

```
for i in range(0, len(s), 2):
    print(s[i])
```

Comparing Strings

```
>>> s1 = "green"
>>> s2 = "glow"
>>> s1 == s2
False
>>> s1 != s2
True
>>> s1 > s2
True
>>> s1 >= s2
True
>>> s1 < s2
False
>>> s1 <= s2
False
```

Testing Characters in a String

str	
isalnum(): bool	Return True if all characters in this string are alphanumeric and there is at least one character.
isalpha(): bool	Return True if all characters in this string are alphabetic and there is at least one character.
isdigit(): bool	Return True if this string contains only number characters.
isidentifier(): bool	Return True if this string is a Python identifier.
islower(): bool	Return True if all characters in this string are lowercase letters and there is at least one character.
isupper(): bool	Return True if all characters in this string are uppercase letters and there is at least one character.
isspace(): bool	Return True if this string contains only whitespace characters.

Searching for Substrings

str	
endswith(s1: str): bool	Returns True if the string ends with the substring s1.
startswith(s1: str): bool	Returns True if the string starts with the substring s1.
find(s1): int	Returns the lowest index where s1 starts in this string, or -1 if s1 is not found in this string.
rfind(s1): int	Returns the highest index where s1 starts in this string, or -1 if s1 is not found in this string.
count(subtring): int	Returns the number of non-overlapping occurrences of this substring.

Converting Strings

str	
capitalize(): str	Returns a copy of this string with only the first character capitalized.
lower(): str	Returns a copy of this string with all characters converted to lowercase.
upper(): str	Returns a copy of this string with all characters converted to uppercase.
title(): str	Returns a copy of this string with the first letter capitalized in each word.
swapcase(): str	Returns a copy of this string in which lowercase letters are converted to uppercase and uppercase to lowercase.
replace(old, new): str	Returns a new string that replaces all the occurrence of the old string with a new string.

Stripping Whitespace Characters

str	
lstrip(): str	Returns a string with the leading whitespace characters removed.
rstrip(): str	Returns a string with the trailing whitespace characters removed.
strip(): str	Returns a string with the starting and trailing whitespace characters removed.

Formatting Strings

str
center(width): str
ljust(width): str
rjust(width): str
format(items): str

Returns a copy of this string centered in a field of the given width.

Returns a string left justified in a field of the given width.

Returns a string right justified in a field of the given width.

Formats a string. See Section 3.6.

Problem: Finding Palindromes

- Objective: Checking whether a string is a palindrome: a string that reads the same forward and backward.

Problem: Converting Hex to Decimal

Operator Overloading

Defining methods for operators is called *operator overloading*. *Operator overloading* allows the programmer to use the built-in operators for user-defined methods. These methods are named in a special way for Python to recognize the association.

Operators and Methods

Operator	Method
+	__add__(self, other)
*	__mul__(self, other)
-	__sub__(self, other)
/	__div__(self, other)
%	__mod__(self, other)
<	__lt__(self, other)
<=	__le__(self, other)
==	__eq__(self, other)
!=	__ne__(self, other)
>	__gt__(self, other)
>=	__ge__(self, other)
[index]	__getitem__(self, index)
in	__contains__(self, value)
len	__len__(self)

The `Rational` Class

Rational	
-numerator: int	The numerator of this rational number.
-denominator: int	The denominator of this rational number.
Rational(numerator = 0: int, denominator = 1: int)	Creates a rational number with specified numerator (default 0) and denominator (default 1).
__add__(secondRational: Rational): Rational	Returns the addition of this rational with another.
__sub__(secondRational: Rational): Rational	Returns the subtraction of this rational with another.
__mul__(secondRational: Rational): Rational	Returns the multiplication of this rational with another.
__div__(secondRational: Rational): Rational	Returns the division of this rational with another.
__lt__ (secondRational: Rational): bool	Compare this rational number with another.
Also __le__, __eq__, __ne__, __gt__, __ge__ are supported	
__int__(): int	Returns the numerator / denominator as an integer.
__float__(): float	Returns the numerator / denominator.
__str()__: str	Returns a string in the form "numerator / denominator." Returns numerator if denominator is 1.
__getitem(i)__	[0] for numerator and [1] for denominator.

Rational

TestRationalClass

Run