Работа со строками

Строки

Строка представляет последовательность символов в кодировке Unicode, заключенных в кавычки. Причем для определения строк Python позволяет использовать как одинарные, так и двойные кавычки

```
message = "Hello World!"

print(message)  # Hello World!
```

Строки

Если строка длинная, ее можно разбить на части и разместить их на разных строках кода. В этом случае вся строка заключается в круглые скобки, а ее отдельные части - в кавычки:

```
text = ("Laudate omnes gentes laudate "
        "Magnificat in secula ")
print(text)
```

Строки

Многострочный текст заключается в тройные двойные или одинарные кавычки:

```
text = '''Laudate omnes gentes laudate
Magnificat in secula
Et anima mea laudate
Magnificat in secula
'''
print(text)
```

Управляющие последовательности в строке

\ позволяет добавить внутрь строки слеш

\' позволяет добавить внутрь строки одинарную кавычку

\" позволяет добавить внутрь строки двойную кавычку

\n осуществляет переход на новую строку

\t добавляет табуляцию (4 отступа)

Управляющие последовательности в строке

Пример:

```
text = "Message:\n\"Hello World\""
print(text)
```

Консольный вывод:

```
Message:
"Hello World"
```

Управляющие последовательности в строке

Пример:

```
path = "C:\python\name.txt"
print(path)
```

Консольный вывод:

```
C:\python
ame.txt
```

Вставка значений в строку

Для этого внутри строки переменные размещаются в фигурных скобках {}, а перед всей строкой ставится символ f:

```
userName = "Tom"
userAge = 37
user = f"name: {userName}  age: {userAge}"
print(user)   # name: Tom  age: 37
```

Обращение к символам строки

Для обращения к отдельным символам строки по индексу в квадратных скобках

```
string = "hello world"
letter = string[0]  # h
print(c0)
letter = string[6]  # w
print(c6)
```

Обращение к символам строки

Чтобы получить доступ к символам, начиная с конца строки, можно использовать отрицательные индексы. Так, индекс -1 будет представлять последний символ, а -2 - предпоследний символ и так

```
string = "hello world"

letter = string[-1]  # d

letter = string[-5]  # w
```

Строки

Строка - это неизменяемый (immutable) тип, поэтому если мы попробуем изменить какой-то отдельный символ строки, то мы получим ошибку, как в следующем случае:

```
string = "hello world"
string[1] = "R" #Ошибка
```

Мы можем только полностью переустановить значение строки, присвоив ей другое значение.

Перебор строки

С помощью цикла for можно перебрать все символы строки:

```
string = "hello world"
for char in string:
    print(char)
```

Получение подстроки

string[:end]: извлекается последовательность символов начиная с 0-го индекса по индекс end (не включая)

string[start:end]: извлекается последовательность символов начиная с индекса start по индекс end

string[start:end:step]: извлекается последовательность символов начиная с индекса start по индекс end через шаг step

Перебор строки

```
string = "hello world"
# с 0 до 5 индекса
sub_string1 = string[:5]
print(sub_string1)     # hello
# со 2 до 5 индекса
sub_string2 = string[2:5]
print(sub_string2)     # llo
# с 2 по 9 индекса через один символ
sub_string3 = string[2:9:2]
print(sub_string3)     # lowr
```

Объединение строк

Для объединения строк применяется операция сложения:

```
name = "Tom"
surname = "Smith"
fullname = name + " " + surname
print(fullname)  # Tom Smith
```

Объединение строк

В случае, если необходимо сложить число и строку, нужно привести число к строке с помощью функции

```
name = "Tom"
age = 33
info = "Name: " + name + " Age: " + str(age)
print(info)  # Name: Tom Age: 33
```

Сравнение строк

Сравнение производится в лексикографическом порядке. При сравнении строк принимается во внимание символы и их регистр

```
str1 = "1a"
str2 = "aa"
str3 = "Aa"
print(str1 > str2)  # False, так как первый символ в str1 - цифра
print(str2 > str3)  # True, так как первый символ в str2 - в нижнем регистре
```

Регистр

Функция lower() приводит строку к нижнему регистру, а функция upper() - к верхнему.

```
str1 = "Tom"
str2 = "tom"
print(str1 == str2)  # False - строки не равны
print(str1.lower() == str2.lower())  # True
```

Функции ord и len

Поскольку строка содержит символы Unicode, то с помощью функции ord() есть возможность получить числовое значение для символа в кодировке Unicode:

```
print(ord("A"))    # 65
```

Функции ord и len

Для получения длины строки можно использовать функцию len():

```
string = "hello world"
length = len(string)
print(length)   # 11
```

Поиск в строке

С помощью выражения term in string можно найти подстроку term в строке string. Если подстрока найдена, то выражение вернет значение True, иначе возвращается значение False:

```
text = "hello world"
exist = "hello" in text
print(exist)    # True
exist = "sword" in text
print(exist)    # False
```

Поиск в строке

Соответственно с помощью операторов not in можно проверить отсутствие подстроки в строке:

```
text = "hello world"
print("hello" not in text)    # False
print("sword" not in text)    # True
```

```python
lang = input("What's the programming language you want to learn? ")

match lang:
    case "JavaScript":
        print("You can become a web developer.")

    case "Python":
        print("You can become a Data Scientist")

    case "PHP":
        print("You can become a backend developer")

    case "Solidity":
        print("You can become a Blockchain developer")

    case "Java":
        print("You can become a mobile app developer")
    case _:
        print("The language doesn't matter, what matters is solving problems.")
```