Python - Strings

A **string** in Python is a sequence of characters enclosed within:

- **Single quotes** (`'Hello'`)
- **Double quotes** (`"Hello"`)
- **Triple quotes** (`'''Hello'''` or `"""Hello"""` - for multi-line strings)

```
str1 = 'Hello'
str2 = "World"
str3 = '''Python is awesome'''
```

Strings are **immutable**, meaning they cannot be changed after creation.

Python - Slicing Strings

Slicing is extracting parts of a string using **indexing**.

Syntax:

```
string[start:end:step]
```

In [1]:
```
s = "Python"
print(s[0:4])   # 'Pyth' (characters from index 0 to 3)
print(s[:4])    # 'Pyth' (start is 0 by default)
print(s[2:])    # 'thon' (go from index 2 to end)
print(s[-3:])   # 'hon' (negative indexing)
print(s[::2])   # 'Pto' (every second character)
print(s[::-1])  # 'nohtyP' (reversed string)
```

```
Pyth
Pyth
thon
hon
Pto
nohtyP
```

Python - Modify Strings

Python provides several methods to modify strings:

In [2]:
```
s = " hello world "

print(s.upper())    # ' HELLO WORLD '
print(s.lower())    # ' hello world '
print(s.strip())    # 'hello world' (removes leading & trailing spaces)
print(s.replace("world", "Python"))  # ' hello Python '
print(s.title())    # ' Hello World '
print(s.capitalize())  # ' hello world ' → ' hello world'
```

```
 HELLO WORLD
 hello world
hello world
 hello Python
 Hello World
 hello world
```

Python - String Concatenation

String concatenation means **joining** two or more strings.

In [3]:
```python
s1 = "Hello"
s2 = "World"

# Using +
print(s1 + " " + s2)  # 'Hello World'

# Using f-string
print(f"{s1} {s2}")  # 'Hello World'

# Using join()
print(" ".join([s1, s2]))  # 'Hello World'
```

```
Hello World
Hello World
Hello World
```

Python - String Formatting

Python provides multiple ways to format strings.

1. Using `format()`

In [4]:
```python
name = "Alice"
age = 25
print("My name is {} and I am {} years old.".format(name, age))
```

```
My name is Alice and I am 25 years old.
```

2. Using f-strings (Python 3.6+)

In [5]:
```python
print(f"My name is {name} and I am {age} years old.")
```

```
My name is Alice and I am 25 years old.
```

3. Using `%` Formatting (Old Style)

In [6]:
```python
print("My name is %s and I am %d years old." % (name, age))
```

```
My name is Alice and I am 25 years old.
```

4. Using `.format()` with indexes

In [7]:
```python
print("I love {1}, {0}, and {2}.".format("C", "Python", "Java"))
```

```
I love Python, C, and Java.
```

Python - Escape Characters

Escape characters allow you to insert characters that are difficult to type.

Escape Sequence	Meaning
`\n`	Newline
`\t`	Tab
`\\`	Backslash
`\'`	Single quote
`\"`	Double quote
`\r`	Carriage return
`\b`	Backspace

In [8]:
```
print("Hello\nWorld")  # Newline
print("Python\tRocks")  # Tab
print("She said, \"Python is amazing!\"")  # Quotes
```

```
Hello
World
Python  Rocks
She said, "Python is amazing!"
```

Python - String Methods

Python provides several built-in methods for string manipulation.

In [9]:
```
s = " Hello, Python! "

print(s.upper())     # ' HELLO, PYTHON! '
print(s.lower())     # ' hello, python! '
print(s.strip())     # 'Hello, Python!' (removes spaces)
print(s.split(","))  # [' Hello', ' Python!'] (splitting)
print(s.replace("Python", "World"))  # ' Hello, World! '
print(s.startswith("H"))  # False (because of space)
print(s.endswith("!"))  # True
```

```
 HELLO, PYTHON!
 hello, python!
Hello, Python!
[' Hello', ' Python! ']
 Hello, World!
False
False
```

Method	Description
`strip()`	Removes spaces at the beginning and end
`replace(old, new)`	Replaces a substring
`split(separator)`	Splits string into a list
`join(iterable)`	Joins elements of a list into a string

Method	Description
`find(substring)`	Returns the first occurrence index
`count(substring)`	Counts occurrences of a substring

Key Notes on Python Strings

1. **Immutable** – Strings **cannot** be changed after creation.
2. **Defined using quotes** – Single (`'text'`), double (`"text"`), or triple (`'''text'''`).
3. **Indexing** – Access characters using **positive** (`s[0]`) & **negative** (`s[-1]`) indexing.
4. **Slicing** – Extract substrings using `s[start:end]` .
5. **Concatenation** – Combine strings using `+` .
6. **Repetition** – Repeat a string using `*` .
7. **String Formatting** – Use `.format()` , `f"{variable}"` , or `%` formatting.
8. **Escape Characters** – `\n` (newline), `\t` (tab), `\\` (backslash), `\'` (single quote), `\"` (double quote).
9. **String Methods** – `strip()` , `replace()` , `split()` , `join()` , `find()` , `count()` , `upper()` , `lower()` , etc.

Python - Lists

Python **lists** are:

- ✅ **Ordered**
- ✅ **Mutable**
- ✅ **Allow duplicates**
- ✅ **Store different data types**
- ✅ **Support indexing, slicing, and iteration**
- ✅ **Highly flexible and widely used**

Creating Lists with Specific Values

```
marks = [85, 90, 78, 92, 88]  # List of student marks
fruits = ["Mango", "Apple", "Banana", "Orange"]  # List of fruits
prices = [199.99, 499.50, 299.75, 599.00]  # List of product prices
```

Python - Access List Items

You can access elements using **indexing** and **negative indexing**.

In [10]:
```
fruits = ["Mango", "Apple", "Banana", "Orange"]

print(fruits[0])   # Output: Mango
print(fruits[2])   # Output: Banana
print(fruits[-1])  # Output: Orange
print(fruits[-3])  # Output: Apple
```

```
Mango
Banana
Orange
Apple
```

Slicing Lists with Specific Values

In [11]:
```python
marks = [85, 90, 78, 92, 88]

print(marks[1:4])   # Output: [90, 78, 92] (Index 1 to 3)
print(marks[:3])    # Output: [85, 90, 78] (First three marks)
print(marks[2:])    # Output: [78, 92, 88] (From index 2 to end)
print(marks[::-1])  # Output: [88, 92, 78, 90, 85] (Reverse list)
```

```
[90, 78, 92]
[85, 90, 78]
[78, 92, 88]
[88, 92, 78, 90, 85]
```

Python - Change List Items

In [12]:
```python
marks = [85, 90, 78, 92, 88]
marks[2] = 80  # Change 78 to 80
print(marks)   # Output: [85, 90, 80, 92, 88]
```

```
[85, 90, 80, 92, 88]
```

Modifying a Range of Items

In [13]:
```python
prices = [199, 499, 299, 599]
prices[1:3] = [450, 280]
print(prices)  # Output: [199, 450, 280, 599]
```

```
[199, 450, 280, 599]
```

Python - Add List Items

1. Using `append()` to Add a Single Item

In [14]:
```python
fruits = ["Mango", "Apple", "Banana"]
fruits.append("Pineapple")
print(fruits)  # Output: ['Mango', 'Apple', 'Banana', 'Pineapple']
```

```
['Mango', 'Apple', 'Banana', 'Pineapple']
```

2. Using `insert()` to Add at a Specific Position

In [15]:
```python
marks = [85, 90, 92]
marks.insert(1, 88)  # Insert 88 at index 1
print(marks)  # Output: [85, 88, 90, 92]
```

```
[85, 88, 90, 92]
```

3. Using `extend()` to Add Multiple Items

In [16]:
```
numbers = [1, 2, 3]
more_numbers = [4, 5, 6]
numbers.extend(more_numbers)
print(numbers)  # Output: [1, 2, 3, 4, 5, 6]
```

```
[1, 2, 3, 4, 5, 6]
```

Python - Remove List Items

1. Using `remove()` to Remove by Value

In [17]:
```
fruits = ["Mango", "Apple", "Banana", "Orange"]
fruits.remove("Apple")
print(fruits)  # Output: ['Mango', 'Banana', 'Orange']
```

```
['Mango', 'Banana', 'Orange']
```

2. Using `pop()` to Remove by Index

In [18]:
```
prices = [199, 499, 299, 599]
prices.pop(2)  # Removes element at index 2 (299)
print(prices)  # Output: [199, 499, 599]
```

```
[199, 499, 599]
```

3. Using `del` to Delete an Item or the Whole List

In [19]:
```
marks = [85, 90, 78, 92, 88]
del marks[3]  # Deletes 92
print(marks)  # Output: [85, 90, 78, 88]

del marks  # Deletes entire list
```

```
[85, 90, 78, 88]
```

4. Using `clear()` to Empty the List

In [20]:
```
numbers = [1, 2, 3, 4, 5]
numbers.clear()
print(numbers)  # Output: []
```

```
[]
```

Python - Loop Lists

1. Using `for` Loop

In [21]:
```
students = ["John", "Emma", "Sophia", "Mike"]
for student in students:
    print(student)
```

```
John
Emma
Sophia
Mike
```

2. Using `while` Loop

In [22]:
```python
marks = [85, 90, 78, 92, 88]
i = 0
while i < len(marks):
    print(marks[i])
    i += 1
```

```
85
90
78
92
88
```

Python - List Comprehension

In [23]:
```python
numbers = [1, 2, 3, 4, 5]
squares = [num ** 2 for num in numbers]
print(squares)  # Output: [1, 4, 9, 16, 25]
```

```
[1, 4, 9, 16, 25]
```

Python - Sort Lists

1. Sorting Numbers

In [24]:
```python
ages = [25, 19, 42, 30]
ages.sort()
print(ages)  # Output: [19, 25, 30, 42]

ages.sort(reverse=True)
print(ages)  # Output: [42, 30, 25, 19]
```

```
[19, 25, 30, 42]
[42, 30, 25, 19]
```

2. Sorting Strings (Case-Insensitive)

In [25]:
```python
fruits = ["banana", "Apple", "cherry"]
fruits.sort(key=str.lower)
print(fruits)  # Output: ['Apple', 'banana', 'cherry']
```

```
['Apple', 'banana', 'cherry']
```

Python - Copy Lists

In [26]:
```python
original = [10, 20, 30]
copy1 = original.copy()
copy2 = list(original)

print(copy1)  # Output: [10, 20, 30]
print(copy2)  # Output: [10, 20, 30]
```

```
[10, 20, 30]
[10, 20, 30]
```

Python - Join Lists

In [27]:
```python
list1 = ["Red", "Blue"]
list2 = ["Green", "Yellow"]
combined = list1 + list2
print(combined)  # Output: ['Red', 'Blue', 'Green', 'Yellow']
```

```
['Red', 'Blue', 'Green', 'Yellow']
```

Python - List Methods

Method	Description	Example	Output
`append(x)`	Adds `x` at the end of the list	`fruits.append("Peach")`	`['Apple', 'Banana', 'Peach']`
`insert(i, x)`	Inserts `x` at index `i`	`fruits.insert(1, "Grapes")`	`['Apple', 'Grapes', 'Banana']`
`remove(x)`	Removes the first occurrence of `x`	`fruits.remove("Banana")`	`['Apple', 'Peach']`
`pop(i)`	Removes element at index `i`	`fruits.pop(1)`	`['Apple', 'Peach']`
`sort()`	Sorts the list	`numbers.sort()`	`[1, 2, 3, 4, 5]`
`reverse()`	Reverses the list	`numbers.reverse()`	`[5, 4, 3, 2, 1]`

Python - Tuples

Python Tuples are:

- ✅ **Ordered**
- ✅ **Immutable** (Cannot be changed after creation)
- ✅ **Allow duplicates**
- ✅ **Store different data types**
- ✅ **Support indexing, slicing, and iteration**
- ✅ **Memory efficient and faster than lists**

Creating Tuples with Specific Values

```python
marks = (85, 90, 78, 92, 88)  # Tuple of student marks
fruits = ("Mango", "Apple", "Banana", "Orange")  # Tuple of fruits
prices = (199.99, 499.50, 299.75, 599.00)  # Tuple of product prices
```

Python - Creating Tuples

You can create a tuple using **parentheses** `()` or the `tuple()` constructor.

In [28]:
```
# Creating a tuple
fruits = ("Apple", "Banana", "Cherry")
print(fruits)  # ('Apple', 'Banana', 'Cherry')

# Tuple with different data types
mixed_tuple = (1, "Hello", 3.14, True)
print(mixed_tuple)  # (1, 'Hello', 3.14, True)

# Single-element tuple (comma is needed!)
single_tuple = ("Python",)
print(type(single_tuple))  # <class 'tuple'>

# Without the comma, it's just a string
not_a_tuple = ("Python")
print(type(not_a_tuple))  # <class 'str'>
```

```
('Apple', 'Banana', 'Cherry')
(1, 'Hello', 3.14, True)
<class 'tuple'>
<class 'str'>
```

Python - Access Tuple Items

You can **access** tuple elements using **indexing** and **negative indexing**.

In [29]:
```
numbers = (10, 20, 30, 40, 50)

# Accessing by index
print(numbers[1])  # 20

# Negative indexing
print(numbers[-1])  # 50 (last item)
print(numbers[-3])  # 30
```

```
20
50
30
```

Python - Update Tuples

Since tuples are **immutable**, you **cannot** modify elements directly. However, you can:

1. Convert the tuple to a **list**, update it, and convert it back.
2. **Concatenate tuples** to create a new one.

In [30]:
```
# Convert to list and modify
colors = ("Red", "Green", "Blue")
colors_list = list(colors)
colors_list[1] = "Yellow"
colors = tuple(colors_list)
print(colors)  # ('Red', 'Yellow', 'Blue')

# Concatenation to add elements
colors = colors + ("Purple",)
print(colors)  # ('Red', 'Yellow', 'Blue', 'Purple')
```

```
('Red', 'Yellow', 'Blue')
('Red', 'Yellow', 'Blue', 'Purple')
```

Python - Unpack Tuples

Tuple unpacking allows you to **assign elements to multiple variables**.

In [31]:
```
person = ("Alice", 25, "Engineer")

name, age, job = person
print(name)  # Alice
print(age)   # 25
print(job)   # Engineer

# Using * to collect remaining values
numbers = (1, 2, 3, 4, 5)
a, *b, c = numbers
print(a)  # 1
print(b)  # [2, 3, 4]
print(c)  # 5
```

```
Alice
25
Engineer
1
[2, 3, 4]
5
```

Python - Loop Tuples

You can loop through tuples using **for loops**.

In [32]:
```
fruits = ("Apple", "Banana", "Cherry")

for fruit in fruits:
    print(fruit)
```

```
Apple
Banana
Cherry
```

Python - Join Tuples

Tuples can be combined using `+` **(concatenation)** and `*` **(repetition)**.

In [33]:
```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)

# Joining tuples
merged_tuple = tuple1 + tuple2
print(merged_tuple)  # (1, 2, 3, 4, 5, 6)

# Repeating elements
repeated_tuple = ("Hello",) * 3
print(repeated_tuple)  # ('Hello', 'Hello', 'Hello')
```

```
(1, 2, 3, 4, 5, 6)
('Hello', 'Hello', 'Hello')
```

Python - Tuple Methods

Tuples have a few built-in methods:

Method	Description	Example	Output
`count(x)`	Counts occurrences of `x` in tuple	`(1, 2, 3, 1, 1).count(1)`	`3`
`index(x)`	Returns index of first occurrence	`("a", "b", "c").index("b")`	`1`

In [34]:
```
numbers = (1, 2, 3, 1, 1)
print(numbers.count(1))  # 3

letters = ("a", "b", "c", "b")
print(letters.index("b"))  # 1
```

```
3
1
```

Python - Sets

A **set** is an **unordered, mutable collection** of **unique elements** in Python.

✅ **Unordered** – No specific order of elements
✅ **Unique** – No duplicate elements
✅ **Mutable** – Can add/remove elements
✅ **Set operations** – Union, Intersection, Difference, etc.

Creating a Set

```
fruits = {"Apple", "Banana", "Mango", "Orange"}
numbers = {1, 2, 3, 4, 5}
print(fruits)  # Output may vary due to unordered nature
```

Python - Access Set Items

- **Sets do not support indexing** since they are unordered.
- You can use a loop or check for element existence using `in`.

In [35]:
```
fruits = {"Apple", "Banana", "Mango"}
print("Apple" in fruits)  # True
print("Grapes" in fruits)  # False
```

```
True
False
```

Python - Add Set Items

- Use `.add()` to add a single element.
- Use `.update()` to add multiple elements.

In [36]:
```
fruits = {"Apple", "Banana"}
fruits.add("Mango")
print(fruits)  # {'Apple', 'Banana', 'Mango'}

fruits.update(["Orange", "Grapes"])
print(fruits)  # {'Apple', 'Banana', 'Mango', 'Orange', 'Grapes'}
```

```
{'Apple', 'Mango', 'Banana'}
{'Grapes', 'Mango', 'Orange', 'Apple', 'Banana'}
```

Python - Remove Set Items

- Use `.remove(x)` to remove an element (**raises an error if not found**).
- Use `.discard(x)` to remove an element (**doesn't raise an error**).
- Use `.pop()` to remove a **random element**.

In [37]:
```
fruits = {"Apple", "Banana", "Mango"}
fruits.remove("Banana")  # Removes 'Banana'
fruits.discard("Grapes")  # No error even if 'Grapes' isn't in set
print(fruits)

fruits.pop()  # Removes a random element
print(fruits)
```

```
{'Apple', 'Mango'}
{'Mango'}
```

Python - Loop Sets

- Use a `for` loop to iterate over a set.

In [38]:
```
fruits = {"Apple", "Banana", "Mango"}
for fruit in fruits:
    print(fruit)
```

```
Apple
Mango
Banana
```

Python - Join Sets

- Use `.union()` to combine sets (**returns a new set**).
- Use `.update()` to add elements from another set (**modifies the original set**).

In [39]:
```
set1 = {1, 2, 3}
set2 = {3, 4, 5}

union_set = set1.union(set2)
print(union_set)  # {1, 2, 3, 4, 5}

set1.update(set2)
print(set1)  # {1, 2, 3, 4, 5}
```

```
{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5}
```

Python - Copy Sets

- Use `.copy()` to create a shallow copy.

In [40]:
```
fruits = {"Apple", "Banana", "Mango"}
new_fruits = fruits.copy()
```

```
print(new_fruits)  # {'Apple', 'Banana', 'Mango'}
```

```
{'Apple', 'Mango', 'Banana'}
```

Python - Set Operators

Operator	Description	Example	Output		
`	`or` .union()`	Combines sets	`{1, 2}	{2, 3}`	`{1, 2, 3}`
`&` or `.intersection()`	Common elements	`{1, 2} & {2, 3}`	`{2}`		
`-` or `.difference()`	Elements in first but not second	`{1, 2} - {2, 3}`	`{1}`		
`^` or `.symmetric_difference()`	Elements in either set, but not both	`{1, 2} ^ {2, 3}`	`{1, 3}`		

Python - Set Methods

Method	Description	Example	Output
`add(x)`	Adds `x` to the set	`fruits.add("Grapes")`	`{"Apple", "Banana", "Mango", "Grapes"}`
`remove(x)`	Removes `x`, raises an error if not found	`fruits.remove("Banana")`	`{"Apple", "Mango"}`
`discard(x)`	Removes `x`, no error if not found	`fruits.discard("Grapes")`	`{"Apple", "Mango"}`
`pop()`	Removes a random element	`fruits.pop()`	Random element removed
`clear()`	Removes all elements	`fruits.clear()`	`{}`

Python - Dictionaries

A **dictionary** in Python is an **unordered, mutable collection** of **key-value pairs**.

✅ **Unordered** – Elements are not stored in a fixed order
✅ **Mutable** – Can add, modify, or remove elements
✅ **Unique keys** – Each key must be unique
✅ **Key-value pairs** – Data is stored in `{key: value}` format

Creating a Dictionary

```
student = {"name": "John", "age": 21, "course": "CS"}
print(student)  # {'name': 'John', 'age': 21, 'course': 'CS'}
```

Python - Access Dictionary Items

- Use **square brackets** `[]` or `.get()` to access values.

In [41]:
```
student = {"name": "John", "age": 21, "course": "CS"}

# Accessing using key
print(student["name"])  # Output: John

# Accessing using get()
print(student.get("age"))
```

```
John
21
```

- **Advantage of** `.get()`: If the key doesn't exist, it **doesn't raise an error**.

In [42]:
```
print(student.get("grade", "Not Found"))
```

```
Not Found
```

Python - Change Dictionary Items

- Update the value of a specific key.

In [43]:
```
student["age"] = 22
print(student)
```

```
{'name': 'John', 'age': 22, 'course': 'CS'}
```

- Use `.update()` to update multiple keys.

In [44]:
```
student.update({"age": 23, "grade": "A"})
print(student)
```

```
{'name': 'John', 'age': 23, 'course': 'CS', 'grade': 'A'}
```

Python - Add Dictionary Items

- Add a new key-value pair.

In [45]:
```
student["city"] = "New York"
print(student)
```

```
{'name': 'John', 'age': 23, 'course': 'CS', 'grade': 'A', 'city': 'New York'}
```

Python - Remove Dictionary Items

- Use `del` or `.pop()` to remove items.

In [46]:
```
# Using del
del student["course"]
print(student)  # {'name': 'John', 'age': 23, 'grade': 'A', 'city': 'New York'}

# Using pop()
age = student.pop("age")
```

```
print(age)  # Output: 23
print(student)  # {'name': 'John', 'grade': 'A', 'city': 'New York'}
```

```
{'name': 'John', 'age': 23, 'grade': 'A', 'city': 'New York'}
23
{'name': 'John', 'grade': 'A', 'city': 'New York'}
```

- Use `.popitem()` to remove the **last inserted** item.

In [47]:
```
student.popitem()
print(student)
```

```
{'name': 'John', 'grade': 'A'}
```

- Use `.clear()` to empty the dictionary.

In [48]:
```
student.clear()
print(student)
```

```
{}
```

Python - Dictionary View Objects

Dictionaries have **special view objects** for keys, values, and key-value pairs.

In [49]:
```
student = {"name": "John", "age": 21, "course": "CS"}

print(student.keys())  # dict_keys(['name', 'age', 'course'])
print(student.values())  # dict_values(['John', 21, 'CS'])
print(student.items())  # dict_items([('name', 'John'), ('age', 21), ('course', 'CS
```

```
dict_keys(['name', 'age', 'course'])
dict_values(['John', 21, 'CS'])
dict_items([('name', 'John'), ('age', 21), ('course', 'CS')])
```

Python - Loop Dictionaries

- Iterate over keys, values, or both.

In [50]:
```
student = {"name": "John", "age": 21, "course": "CS"}

# Loop through keys
for key in student:
    print(key, ":", student[key])

# Loop through values
for value in student.values():
    print(value)

# Loop through key-value pairs
for key, value in student.items():
    print(f"{key}: {value}")
```

```
name : John
age : 21
course : CS
John
21
CS
name: John
age: 21
course: CS
```

Python - Copy Dictionaries

- Use `.copy()` to create a **shallow copy**.

In [51]:
```python
student = {"name": "John", "age": 21}
student_copy = student.copy()
print(student_copy)  # {'name': 'John', 'age': 21}
```

```
{'name': 'John', 'age': 21}
```

- Using `dict()` to create a copy.

In [52]:
```python
student_copy = dict(student)
print(student_copy)  # {'name': 'John', 'age': 21}
```

```
{'name': 'John', 'age': 21}
```

Python - Nested Dictionaries

- A dictionary inside another dictionary.

In [53]:
```python
students = {
    "student1": {"name": "John", "age": 21},
    "student2": {"name": "Jane", "age": 22},
}

print(students["student1"]["name"])  # Output: John
```

```
John
```

Python - Dictionary Methods

Method	Description	Example	Output
`get(key, default)`	Returns the value of `key`, or default if not found	`student.get("grade", "Not Found")`	`"Not Found"`
`update(dict)`	Updates dictionary with another dictionary	`student.update({"age": 23})`	`{'name': 'John', 'age': 23}`
`pop(key)`	Removes key and returns value	`student.pop("age")`	`21`
`popitem()`	Removes last inserted item	`student.popitem()`	`("course", "CS")`

Method	Description	Example	Output
`clear()`	Removes all items	`student.clear()`	`{}`
`keys()`	Returns dictionary keys	`student.keys()`	`dict_keys(['name', 'age'])`
`values()`	Returns dictionary values	`student.values()`	`dict_values(['John', 21])`
`items()`	Returns key-value pairs	`student.items()`	`dict_items([('name', 'John'), ('age', 21)])`
`copy()`	Returns a shallow copy	`student.copy()`	`{'name': 'John', 'age': 21}`