## When To Use

- When you need an input box instead of a selector.
- When you need input suggestions or helping text.

The differences with Select are:

- AutoComplete is an input box with text hints, and users can type freely. The keyword is aiding **input**.
- Select is selecting among given choices. The keyword is **select**.

## Examples

### Basic Usage

```
import React, { useState } from 'react';
import { AutoComplete } from 'antd';
import type { AutoCompleteProps } from 'antd';

const mockVal = (str: string, repeat = 1) => ({
  value: str.repeat(repeat),
});

const App: React.FC = () => {
  const [value, setValue] = useState('');
  const [options, setOptions] = useState<AutoCompleteProps['options']>([]);
  const [anotherOptions, setAnotherOptions] =
useState<AutoCompleteProps['options']>([]);

  const getPanelValue = (searchText: string) =>
    !searchText ? [] : [mockVal(searchText), mockVal(searchText, 2),
mockVal(searchText, 3)];

  const onSelect = (data: string) => {
    console.log('onSelect', data);
  };

  const onChange = (data: string) => {
    setValue(data);
  };

  return (
    <>
      <AutoComplete
        options={options}
        style={{ width: 200 }}
        onSelect={onSelect}
        onSearch={(text) => setOptions(getPanelValue(text))}
        placeholder="input here"
      />
      <br />
      <br />
      <AutoComplete
        value={value}
        options={anotherOptions}
        style={{ width: 200 }}
        onSelect={onSelect}
        onSearch={(text) => setAnotherOptions(getPanelValue(text))}
        onChange={onChange}
        placeholder="control mode"
      />
    </>
  );
};

export default App;
```

**Customized**

```
import React from 'react';
import { AutoComplete } from 'antd';
import type { AutoCompleteProps } from 'antd';

const App: React.FC = () => {
  const [options, setOptions] =
React.useState<AutoCompleteProps['options']>([]);
  const handleSearch = (value: string) => {
    setOptions(() => {
      if (!value || value.includes('@')) {
        return [];
      }
      return ['gmail.com', '163.com', 'qq.com'].map((domain) => ({
        label: `${value}@${domain}`,
        value: `${value}@${domain}`,
      }));
    });
  };
  return (
    <AutoComplete
      style={{ width: 200 }}
      onSearch={handleSearch}
      placeholder="input here"
      options={options}
    />
  );
};

export default App;
```

**Customize Input Component**

```
import React, { useState } from 'react';
import { AutoComplete, Input } from 'antd';
import type { AutoCompleteProps } from 'antd';

const { TextArea } = Input;

const App: React.FC = () => {
  const [options, setOptions] = useState<AutoCompleteProps['options']>([]);

  const handleSearch = (value: string) => {
    setOptions(
      !value ? [] : [{ value }, { value: value + value }, { value: value +
value + value }],
    );
  };

  const handleKeyPress = (ev: React.KeyboardEvent<HTMLTextAreaElement>) =>
{
    console.log('handleKeyPress', ev);
  };

  const onSelect = (value: string) => {
    console.log('onSelect', value);
  };

  return (
    <AutoComplete
      options={options}
      style={{ width: 200 }}
      onSelect={onSelect}
      onSearch={handleSearch}
    >
      <TextArea
        placeholder="input here"
        className="custom"
        style={{ height: 50 }}
        onKeyPress={handleKeyPress}
      />
    </AutoComplete>
  );
};

export default App;
```

**Non-case-sensitive AutoComplete**

```
import React from 'react';
import { AutoComplete } from 'antd';

const options = [
  { value: 'Burns Bay Road' },
  { value: 'Downing Street' },
  { value: 'Wall Street' },
];

const App: React.FC = () => (
  <AutoComplete
    style={{ width: 200 }}
    options={options}
    placeholder="try to type `b`"
    filterOption={(inputValue, option) =>
      option!.value.toUpperCase().indexOf(inputValue.toUpperCase()) !== -1
    }
  />
);

export default App;
```

**Lookup-Patterns - Certain Category**

```
import React from 'react';
import { UserOutlined } from '@ant-design/icons';
import { AutoComplete, Flex, Input } from 'antd';

const Title: React.FC<Readonly<{ title?: string }>> = (props) => (
  <Flex align="center" justify="space-between">
    {props.title}
    <a href="https://www.google.com/search?q=antd" target="_blank"
rel="noopener noreferrer">
      more
    </a>
  </Flex>
);
```

```
const renderItem = (title: string, count: number) => ({
  value: title,
  label: (
    <Flex align="center" justify="space-between">
      {title}
      <span>
        <UserOutlined /> {count}
      </span>
    </Flex>
  ),
});

const options = [
  {
    label: <Title title="Libraries" />,
    options: [renderItem('AntDesign', 10000), renderItem('AntDesign UI',
10600)],
  },
  {
    label: <Title title="Solutions" />,
    options: [renderItem('AntDesign UI FAQ', 60100), renderItem('AntDesign
FAQ', 30010)],
  },
  {
    label: <Title title="Articles" />,
    options: [renderItem('AntDesign design language', 100000)],
  },
];

const App: React.FC = () => (
  <AutoComplete
    popupClassName="certain-category-search-dropdown"
    popupMatchSelectWidth={500}
    style={{ width: 250 }}
    options={options}
    size="large"
  >
    <Input.Search size="large" placeholder="input here" />
  </AutoComplete>
);

export default App;
```

**Lookup-Patterns - Uncertain Category**

```
import React, { useState } from 'react';
import { AutoComplete, Input } from 'antd';
import type { AutoCompleteProps } from 'antd';

const getRandomInt = (max: number, min = 0) => Math.floor(Math.random() *
(max - min + 1)) + min;

const searchResult = (query: string) =>
  Array.from({ length: getRandomInt(5) })
    .join('.')
    .split('.')
    .map((_, idx) => {
      const category = `${query}${idx}`;
      return {
        value: category,
        label: (
          <div
            style={{
              display: 'flex',
              justifyContent: 'space-between',
            }}
          >
            <span>
              Found {query} on{' '}
              <a
                href={`https://s.taobao.com/search?q=${query}`}
                target="_blank"
                rel="noopener noreferrer"
              >
                {category}
              </a>
            </span>
            <span>{getRandomInt(200, 100)} results</span>
          </div>
        ),
      };
    });

const App: React.FC = () => {
  const [options, setOptions] = useState<AutoCompleteProps['options']>([]);

  const handleSearch = (value: string) => {
    setOptions(value ? searchResult(value) : []);
  };

  const onSelect = (value: string) => {
    console.log('onSelect', value);
  };

  return (
    <AutoComplete
      popupMatchSelectWidth={252}
      style={{ width: 300 }}
      options={options}
      onSelect={onSelect}
      onSearch={handleSearch}
      size="large"
    >
      <Input.Search size="large" placeholder="input here" enterButton />
    </AutoComplete>
  );
};

export default App;
```

**Status**

```
import React, { useState } from 'react';
import { AutoComplete, Space } from 'antd';
import type { AutoCompleteProps } from 'antd';

const mockVal = (str: string, repeat = 1) => ({
  value: str.repeat(repeat),
});

const App: React.FC = () => {
  const [options, setOptions] = useState<AutoCompleteProps['options']>([]);
  const [anotherOptions, setAnotherOptions] =
useState<AutoCompleteProps['options']>([]);

  const getPanelValue = (searchText: string) =>
    !searchText ? [] : [mockVal(searchText), mockVal(searchText, 2),
mockVal(searchText, 3)];

  return (
    <Space direction="vertical" style={{ width: '100%' }}>
      <AutoComplete
        options={options}
        onSearch={(text) => setOptions(getPanelValue(text))}
        status="error"
        style={{ width: 200 }}
      />
      <AutoComplete
        options={anotherOptions}
        onSearch={(text) => setAnotherOptions(getPanelValue(text))}
        status="warning"
        style={{ width: 200 }}
      />
    </Space>
  );
};

export default App;
```

### Variants

v5.13.0

```
import React, { useState } from 'react';
import { AutoComplete, Flex } from 'antd';
import type { AutoCompleteProps } from 'antd';

const mockVal = (str: string, repeat = 1) => ({
  value: str.repeat(repeat),
});

const App: React.FC = () => {
  const [options, setOptions] = useState<AutoCompleteProps['options']>([]);

  const getPanelValue = (searchText: string) =>
    !searchText ? [] : [mockVal(searchText), mockVal(searchText, 2),
mockVal(searchText, 3)];

  return (
    <Flex vertical gap={12}>
      <AutoComplete
        options={options}
        style={{ width: 200 }}
        placeholder="Outlined"
        onSearch={(text) => setOptions(getPanelValue(text))}
        onSelect={globalThis.console.log}
      />
      <AutoComplete
        options={options}
        style={{ width: 200 }}
        placeholder="Filled"
        onSearch={(text) => setOptions(getPanelValue(text))}
        onSelect={globalThis.console.log}
        variant="filled"
      />
      <AutoComplete
        options={options}
        style={{ width: 200 }}
        placeholder="Borderless"
        onSearch={(text) => setOptions(getPanelValue(text))}
        onSelect={globalThis.console.log}
        variant="borderless"
      />
    </Flex>
  );
};

export default App;
```

### Customize clear button

```
import React, { useState } from 'react';
import { CloseSquareFilled } from '@ant-design/icons';
import { AutoComplete } from 'antd';
import type { AutoCompleteProps } from 'antd';

const mockVal = (str: string, repeat = 1) => ({
  value: str.repeat(repeat),
});

const App: React.FC = () => {
  const [options, setOptions] = useState<AutoCompleteProps['options']>([]);

  const getPanelValue = (searchText: string) =>
    !searchText ? [] : [mockVal(searchText), mockVal(searchText, 2),
mockVal(searchText, 3)];

  return (
    <>
      <AutoComplete
        options={options}
        style={{ width: 200 }}
        onSearch={(text) => setOptions(getPanelValue(text))}
        placeholder="UnClearable"
        allowClear={false}
      />
      <br />
      <br />
      <AutoComplete
```

```
        options={options}
        style={{ width: 200 }}
        onSearch={(text) => setOptions(getPanelValue(text))}
        placeholder="Customized clear icon"
        allowClear={{ clearIcon: <CloseSquareFilled /> }}
      />
    </>
  );
};

export default App;
```

### Debug in Form

Debug

```
import React from 'react';
import { SearchOutlined } from '@ant-design/icons';
import { AutoComplete, Button, Form, Input, TreeSelect } from 'antd';

const formItemLayout = {
  labelCol: {
    xs: { span: 24 },
    sm: { span: 8 },
  },
  wrapperCol: {
    xs: { span: 24 },
    sm: { span: 16 },
  },
};

const App: React.FC = () => (
  <Form style={{ margin: '0 auto' }} {...formItemLayout}>
    <Form.Item label="单独 AutoComplete">
      <AutoComplete />
    </Form.Item>
    <Form.Item label="单独 TreeSelect">
      <TreeSelect />
    </Form.Item>
    <Form.Item label="添加 Input.Group 正常">
      <Input.Group compact>
        <TreeSelect style={{ width: '30%' }} />
        <AutoComplete />
      </Input.Group>
    </Form.Item>
    <Form.Item label="包含 search 图标正常">
```

```
        <AutoComplete>
          <Input suffix={<SearchOutlined />} />
        </AutoComplete>
      </Form.Item>
      <Form.Item label="同时有 Input.Group 和图标发生移位">
        <Input.Group compact>
          <TreeSelect style={{ width: '30%' }} />
          <AutoComplete>
            <Input suffix={<SearchOutlined />} />
          </AutoComplete>
        </Input.Group>
      </Form.Item>
      <Form.Item label="同时有 Input.Group 和 Search 组件发生移位">
        <Input.Group compact>
          <TreeSelect style={{ width: '30%' }} />
          <AutoComplete>
            <Input.Search />
          </AutoComplete>
        </Input.Group>
      </Form.Item>
      <Form.Item label="Input Group 和 Button 结合">
        <Input.Group compact>
          <TreeSelect style={{ width: '20%' }} />
          <AutoComplete>
            <Input.Search />
          </AutoComplete>
          <Button type="primary" icon={<SearchOutlined />}>
            Search
          </Button>
        </Input.Group>
      </Form.Item>
    </Form>
);

export default App;
```

### AutoComplete and Select

Debug

```
import React from 'react';
import { AutoComplete, Flex, Select } from 'antd';

const AutoCompleteAndSelect = () => {
  return (
    <Flex vertical gap={16}>
```

```
      {(['small', 'middle', 'large'] as const).map((size) => (
        <Flex key={size}>
          <Select
            value="centered"
            size={size}
            style={{ width: 200 }}
            searchValue="centered"
            showSearch
          />
          <AutoComplete value="centered" size={size} style={{ width: 200 }}
/>
        </Flex>
      ))}
    </Flex>
  );
};

export default AutoCompleteAndSelect;
```

### _InternalPanelDoNotUseOrYouWillBeFired

Debug

```
import React from 'react';
import { AutoComplete, Space, Switch } from 'antd';

const { _InternalPanelDoNotUseOrYouWillBeFired: InternalAutoComplete } =
AutoComplete;

const App: React.FC = () => {
  const [open, setOpen] = React.useState(false);

  return (
    <Space direction="vertical" style={{ display: 'flex' }}>
      <Switch checked={open} onChange={() => setOpen(!open)} />
      <InternalAutoComplete
        defaultValue="lucy"
        style={{ width: 120 }}
        open={open}
        options={[
          { label: 'Jack', value: 'jack' },
          { label: 'Lucy', value: 'lucy' },
          { label: 'Disabled', value: 'disabled' },
          { label: 'Bamboo', value: 'bamboo' },
        ]}
      />
    </Space>
  );
};

export default App;
```

## API

Common props ref：[Common props]()

| Property | Description | Type | |
|---|---|---|---|
| allowClear | Show clear button | boolean \| { clearIcon?: ReactNode } | fal |
| autoFocus | If get focus when component mounted | boolean | fal |
| backfill | If backfill selected item the input when using keyboard | boolean | fal |
| children (for customize input element) | Customize input element | HTMLInputElement \| HTMLTextAreaElement \| React.ReactElement<InputProps> | <Ir |
| children (for dataSource) | Data source to auto complete | React.ReactElement<OptionProps> \| Array<React.ReactElement<OptionProps>> | - |
| defaultActiveFirstOption | Whether active first option by default | boolean | tru |
| defaultOpen | Initial open state of dropdown | boolean | - |
| defaultValue | Initial selected option | string | - |

| | | | |
|---|---|---|---|
| disabled | Whether disabled select | boolean | fal |
| dropdownRender | Customize dropdown content | (menus: ReactNode) => ReactNode | - |
| popupClassName | The className of dropdown menu | string | - |
| popupMatchSelectWidth | Determine whether the dropdown menu and the select input are the same width. Default set `min-width` same as input. Will ignore when value less than select width. `false` will disable virtual scroll | boolean \| number | tru |
| filterOption | If true, filter options by input, if function, filter options against it. The function will receive two arguments, `inputValue` and `option`, | boolean \| function(inputValue, option) | tru |

| | | | |
|---|---|---|---|
| | if the function returns true, the option will be included in the filtered set; Otherwise, it will be excluded | | |
| getPopupContainer | Parent node of the dropdown. Default to body, if you encountered positioning problems during scroll, try changing to the scrollable area and position relative to it. [Example]() | function(triggerNode) | () = do |
| notFoundContent | Specify content to show when no result matches | ReactNode | - |
| open | Controlled open state of dropdown | boolean | - |
| options | Select options. Will get better perf than jsx definition | { label, value }[] | - |

| | | | |
|---|---|---|---|
| placeholder | The placeholder of input | string | - |
| status | Set validation status | 'error' \| 'warning' | - |
| size | The size of the input box | `large` \| `middle` \| `small` | - |
| value | Selected option | string | - |
| variant | Variants of input | `outlined` \| `borderless` \| `filled` | `ou` |
| virtual | Disable virtual scroll when set to false | boolean | tru |
| onBlur | Called when leaving the component | function() | - |
| onChange | Called when selecting an option or changing an input value | function(value) | - |
| onDropdownVisibleChange | Call when dropdown open | function(open) | - |
| onFocus | Called when entering the component | function() | - |
| onSearch | Called when searching items | function(value) | - |
| onSelect | Called when a option is selected. param is option's | function(value, option) | - |

|  |  |  |  |
|---|---|---|---|
|  | value and option instance |  |  |
| onClear | Called when clear | function | - |
| onInputKeyDown | Called when key pressed | (event: KeyboardEvent) => void | - |
| onPopupScroll | Called when dropdown scrolls | (event: UIEvent) => void | - |

## Methods

| Name | Description | Version |
|---|---|---|
| blur() | Remove focus |  |
| focus() | Get focus |  |

## Design Token

## FAQ

### Why doesn't the text composition system work well with onSearch in controlled mode?

Please use `onChange` to manage control state. `onSearch` is used for searching input which is not the same as `onChange`. Besides, clicking on the option will not trigger the `onSearch` event.

Related issue: [#18230](#) [#17916](#)

### Why won't a controlled open AutoComplete display a drop-down menu when options are empty?

The AutoComplete component is essentially an extension of the Input form element. When the `options` property is empty, displaying empty text could mislead the user into believing the component is not operational, when in fact they are still able to input text. To avoid confusion, the `open` property will not display the drop-down menu when set to `true` and in combination with an empty `options` property. The `open` property must be used in conjunction with the `options` property.