FT Components

FT components turn Python objects into HTML.

FT, or 'FastTags', are the display components of FastHTML. In fact, the word "components" in the context of FastHTML is often synonymous with **FT**.

For example, when we look at a FastHTML app, in particular the views, as well as various functions and other objects, we see something like the code snippet below. It's the `return` statement that we want to pay attention to:

```python
from fasthtml.common import *

def example():
    # The code below is a set of ft components
    return Div(
            H1("FastHTML APP"),
            P("Let's do this"),
            cls="go"
    )
```

Let's go ahead and call our function and print the result:

```python
example()
```

```html
<div class="go">
  <h1>FastHTML APP</h1>
  <p>Let&#x27;s do this</p>
</div>
```

As you can see, when returned to the user from a Python callable, like a function, the ft components are transformed into their string representations of XML or XML-like content such as HTML. More concisely, *ft turns Python objects into HTML.*

Now that we know what ft components look and behave like we can begin to understand them. At their most fundamental level, ft components:

1. Are Python callables, specifically functions, classes, methods of classes, lambda functions, and anything else called with parenthesis that returns a value.
2. Return a sequence of values which has three elements:
    1. The tag to be generated
    2. The content of the tag, which is a tuple of strings/tuples. If a tuple, it is the three-element structure of an ft component

3. A dictionary of XML attributes and their values

3. FastHTML's default ft components words begin with an uppercase letter. Examples include `Title()`, `Ul()`, and `Div()` Custom components have included things like `BlogPost` and `CityMap`.

How FastHTML names ft components

When it comes to naming ft components, FastHTML appears to break from PEP8. Specifically, PEP8 specifies that when naming variables, functions and instantiated classes we use the `snake_case_pattern`. That is to say, lowercase with words separated by underscores. However, FastHTML uses `PascalCase` for ft components.

There's a couple of reasons for this:

1. ft components can be made from any callable type, so adhering to any one pattern doesn't make much sense
2. It makes for easier reading of FastHTML code, as anything that is PascalCase is probably an ft component

Default FT components

FastHTML has over 150 **FT** components designed to accelerate web development. Most of these mirror HTML tags such as `<div>`, `<p>`, `<a>`, `<title>`, and more. However, there are some extra tags added, including:

- `Titled`, a combination of the `Title()` and `H1()` tags
- `Socials`, renders popular social media tags

The `fasthtml.ft` Namespace

Some people prefer to write code using namespaces while adhering to PEP8. If that's a preference, projects can be coded using the `fasthtml.ft` namespace.

```
from fasthtml import ft

ft.Ul(
    ft.Li("one"),
    ft.Li("two"),
    ft.Li("three")
)
```

```
<ul>
  <li>one</li>
  <li>two</li>
  <li>three</li>
</ul>
```

Attributes

This example demonstrates many important things to know about how ft components handle attributes.

```
1   #| echo: False
2   Label(                                                    1
3       "Choose an option",
4       Select(
5           Option("one", value="1", selected=True),          2
6           Option("two", value="2", selected=False),        3
7           Option("three", value=3),                       4
8           cls="selector",                                5
9           _id="counter",                                6
10          **{'@click':"alert('Clicked');"},            7
11      ),
12      _for="counter",                                       8
13  )
```

1. Line 2 demonstrates that FastHTML appreciates `Label` s surrounding their fields.
2. On line 5, we can see that attributes set to the `boolean` value of `True` are rendered with just the name of the attribute.
3. On line 6, we demonstrate that attributes set to the `boolean` value of `False` do not appear in the rendered output.
4. Line 7 is an example of how integers and other non-string values in the rendered output are converted to strings.
5. Line 8 is where we set the HTML class using the `cls` argument. We use `cls` here as `class` is a reserved word in Python. During the rendering process this will be converted to the word "class".
6. Line 9 demonstrates that any named argument passed into an ft component will have the leading underscore stripped away before rendering. Useful for handling reserved words in Python.
7. On line 10 we have an attribute name that cannot be represented as a python variable. In cases like these, we can use an unpacked `dict` to represent these values.
8. The use of `_for` on line 12 is another demonstration of an argument having the leading underscore stripped during render. We can also use `fr` as that will be expanded to `for`.

This renders the following HTML snippet:

```
Label(
    "Choose an option",
    Select(
        Option("one", value="1", selected=True),
        Option("two", value="2", selected=False),
        Option("three", value=3),  # <4>,
        cls="selector",
        _id="counter",
        **{'@click':"alert('Clicked');"},
    ),
    _for="counter",
)
```

```
<label for="counter">
Choose an option
  <select id="counter" @click="alert(&#x27;Clicked&#x27;);" class="selector" name="c
    <option value="1" selected>one</option>
    <option value="2" >two</option>
    <option value="3">three</option>
  </select>
</label>
```

Defining new ft components

It is possible and sometimes useful to create your own ft components that generate non-standard tags that are not in the FastHTML library. FastHTML supports created and defining those new tags flexibly.

For more information, see the Defining new ft components reference page.

FT components and type hints

If you use type hints, we strongly suggest that FT components be treated as the `Any` type.

The reason is that FastHTML leverages python's dynamic features to a great degree. Especially when it comes to `FT` components, which can evaluate out to be `FT|str|None|tuple` as well as anything that supports the `__ft__`, `__html__`, and `__str__` method. That's enough of the Python stack that assigning anything but `Any` to be the FT type will prove an exercise in frustation.

Report an issue