Table of Contents

json

The json package provides comprehensive JSON handling capabilities, including pa
rsing, stringifying, and type-safe conversion between JSON and other MoonBit dat
a types.

Basic JSON Operations

Parsing and Validating JSON

```
1
2   test "parse and validate jsons" {
3
4     assert_true(@json.valid("{\"key\": 42}"))
5     assert_true(@json.valid("[1, 2, 3]"))
6     assert_true(@json.valid("null"))
7     assert_true(@json.valid("false"))
8
9
10    let json = @json.parse("{\"key\": 42}") catch {
11      (_ : @json.ParseError) => panic()
12
13    }
14
15
16    inspect(
17      json.stringify(indent=2),
18      content={
19        let output =
20          #|{
21          #|  "key": 42
22          #|}
23        output
24      },
25    )
26  }
```

Object Navigation

```

  test "json object navigation" {
    let json = @json.parse(
      "{\"string\":\"hello\",\"number\":42,\"array\":[1,2,3]}",
    )


    let string_opt = json.value("string").unwrap().as_string()
    inspect(
      string_opt,
      content=(
        #|Some("hello")
      ),
    )


    let number_opt = json.value("number").unwrap().as_number()
    inspect(number_opt, content="Some(42)")


    let array_opt = json.value("array").unwrap().as_array()
    inspect(array_opt, content="Some([Number(1), Number(2), Number(3)])")


    inspect(json.value("missing"), content="None")
  }
```

Array Navigation

```

  test "json array navigation" {
    let array = @json.parse("[1,2,3,4,5]")


    let first = array.item(0)
    inspect(first, content="Some(Number(1))")


    let missing = array.item(10)
    inspect(missing, content="None")


    let values = array.as_array().unwrap()
    inspect(
      values.iter(),
      content="[Number(1), Number(2), Number(3), Number(4), Number(5)]",
    )
  }
```

Type-Safe JSON Conversion

From JSON to Native Types

```
1
2   test "json decode" {
3
4     let json_number = (42 : Json)
5     let number : Int = @json.from_json(json_number)
6     inspect(number, content="42")
7
8
9     let json_array = ([1, 2, 3] : Json)
10    let array : Array[Int] = @json.from_json(json_array)
11    inspect(array, content="[1, 2, 3]")
12
13
14    let json_map = ({ "a": 1, "b": 2 } : Json)
15    let map : Map[String, Int] = @json.from_json(json_map)
16    inspect(
17      map,
18      content=(
19        #|{"a": 1, "b": 2}
20      ),
21    )
22  }
```

Error Handling with JSON Path

```
1
2   test "json path" {
3
4     try {
5       let _arr : Array[Int] = @json.from_json(([42, "not a number", 49] :
6       panic()
7     } catch {
8       @json.JsonDecodeError((path, msg)) => {
9         inspect(path, content="$[1]")
10        inspect(msg, content="Int::from_json: expected number")
11      }
12    }
13  }
```

JSON-based Snapshot Testing

@json.inspect() can be used as an alternative to inspect() when a value's To Json implementation is considered a better debugging representation than its Sho w implementation. This is particularly true for deeply-nested data structures.

```
1
2   test "json inspection" {
3     let null = null
4
5
6     let json_value : Json = { "key": "value", "numbers": [1, 2, 3] }
7     @json.inspect(json_value, content={ "key": "value", "numbers": [1, 2,
8
9
10    let json_special = { "null": null, "bool": true }
11    @json.inspect(json_special, content={ "null": null, "bool": true })
12  }
```