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

```

test "parse and validate jsons" {

  assert_true(@json valid("{\"key\": 42}"))
  assert_true(@json valid("[1, 2, 3]"))
  assert_true(@json valid("null"))
  assert_true(@json valid("false"))


  let json = @json parse("{\"key\": 42}") catch {
    (_ : @json ParseError) => panic()

  }


  inspect(
    json stringify(indent=2),
    content={
      let output =
        #|{
        #|  "key": 42
        #|}
      output
    },
  )
}
```

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

```

test "json decode" {

  let json_number = (42 : Json)
  let number : Int = @json from_json(json_number)
  inspect(number, content="42")


  let json_array = ([1, 2, 3] : Json)
  let array : Array[Int] = @json from_json(json_array)
  inspect(array, content="[1, 2, 3]")


  let json_map = ({ "a": 1, "b": 2 } : Json)
  let map : Map[String, Int] = @json from_json(json_map)
  inspect(
    map,
    content=(
      #|{"a": 1, "b": 2}
    ),
  )
}
```

Error Handling with JSON Path

```

test "json path" {

  try {
    let _arr : Array[Int] = @json from_json(([42, "not a number", 49] :
    panic()
  } catch {
    @json JsonDecodeError((path, msg)) => {
      inspect(path, content="$[1]")
      inspect(msg, content="Int::from_json: expected number")
    }
  }
}
```

JSON-based Snapshot Testing

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

```

test "json inspection" {
  let null = null


  let json_value : Json = { "key": "value", "numbers": [1, 2, 3] }
  @json inspect(json_value, content={ "key": "value", "numbers": [1, 2,


  let json_special = { "null": null, "bool": true }
  @json inspect(json_special, content={ "null": null, "bool": true })
}
```