Ballerina CHEATSHEET

Basic syntax

File hello.bal:

```
import ballerina/io;

function main(string... args) {
   io:println("Hello, World!");
}
```

```
> ballerina run hello.bal
Hello, World!
```

Hello World service

```
import ballerina/http;

service<http:Service> hello bind {port:9090} {
  say(endpoint caller, http:Request req) {
      http:Response res = new;
      res.setPayload("Hello world!\n");
      _ = caller->respond(res);
  }
}
```

Variables

```
string name = "Ballerina";
var age = 3;
```

Functions

```
// a simple function
function doIt(int a) returns int {}
var result = doIt(4);

// required and defaultable parameters
function doIt(int a, string op = "inc") {}
doIt(5, op = "dec");

// rest parameter
function doIt(int a, string... names) {}
doIt(3, "a", "b", "c");

// pass an array as the rest parameter
string[] letters = ["a", "b", "c", "d"];
doIt(3, ...letters);
```

Values and Types

Simple Basic Types

Type	Values
`int`	64-bit signed integers
`float`	IEEE 754-2008 64-bit floating point numbers
`boolean`	`true` and `false`
`string`	immutable sequences of zero or more Unicode code points
`()`	`()` or `null` (only for `json`-related contexts)

Arrays/Tuples

```
// unsealed array with infinite size.
int[] a = [1, 2, 3, 4, 5, 6, 7, 8];
a[999] = 100;

// sealed array with predefined size
string[2] b = ["apple", "orange"];

// Tuple
(int, string, int) tuple = (1, "value", 5);
string value = tuple[1];
```

record/map/json types

```
// a simple record
type Person record {
   string name;
   int age;
   map address;
};

Person p = { name:"John", age:50 };

// map
map<string> address = {street:"Palm Grove",
city:"Colombo 03", country:"Sri Lanka"};

// JSON
json info = {name:"John", "age":50, address:
{street:"20 Palm Grove", city:"Colombo"},
contacts:[123, 789]};

// access fields with field-access
p.age = 45;
json j = info.name;


// access fields with index-access
p["age"] = 45;
json j = info["name"];
```

xml type

```
// a simple XML
xml x1 = xml`<name>John</name>`;

// an XML with namespaces
xmlns "http://wso2.com" as ns0;
xml x2 = xml `<name id="123" status="single">
                  <ns0:fname>John</ns0:fname>
                  <ns0:lname>John</ns0:lname>
              </name>`;

// an XML literal with interpolation
string lastName = "Doe";
xml x3 = xml`<lname>{{lastName}}</lname>`;

// concatenating XML
xml x4 = x1 + x2 + x3;

// get children by name
xml fiirstNames1 = x2[ns0:fname];
xml fiirstNames2 = x2["{http://wso2.com}fname"];

// get all the children
xml allChildren = x2.*;

// get an attribute by name
string id = x2@["id"];

// get all the attributes as a map
map attributes = <map> x2@;

// set children
x1.setChildren(x3);
```

table type

```
type Student record {
   int id;
   string name;
   int age;
};

// table constrained with Student record
table<Student> tbStudent = table {
   { primarykey id, name, age },
   [
       { 1, "John",  34 },
       { 1, "Anne",  24 }
   ]
};
```

Ballerina CHEATSHEET

Object type

```
type Person object {
   string name;
   int age;

   // constructor method
   new(name, age) {}

   // member function
   function getName() returns string {
      return name;
   }
};

Person p1 = new ("John", 50);
Person p2 = new Person("Doe", 40);

// invoke member functions
string name = p1.getName();
```

Union type

```
string|int|() value = 5;
value = "foo";
value = ();
```

Optional type

```
map<string>? m;
string? s1 = m.name;

// eliminate nil using but-expression
string s2 = m.name but { () => "N/A" };

// eliminate nil using elvis operator
string s3 = m.name ?: "N/A";
```

function type and closures

```
int a = 2;
var outerFunc = (int x) => int {
  int b = 18;
  function (int) innerFunc = (int y) => () {
      a++;
      b--;
  };
  return b;
};
```

Control Structures

If

```
int value = 10;
if(value > 0) {
   io:println("positive number");
} else if (value < 0) {
   io:println("negative number");
} else {
   io:println("zero");
}
```

Loops

```
// while loop
int i = 0;
while(i < 10) {
   if (i == 5) {
       continue;
   }
   if (i == 7) {
       break;
   }
   i++;
}

// foreach loop
string[] colors= ["red", "blue", "white"];
foreach item in colors {
   io:println(item);
}
```

Type switching - match

```
string|int value = 10;
match value {
   string s => { io:println(s); }
   int i => { io:println(i); }
}
```

Error handling

```
// creating an error.
error err = { message : "error message" };

// two error handling approaches
//  1) return errors
return err;

//  2) throw errors and try/catch
try {
   // some logic
   throw err;
} catch(error er){
```

```
    // error handling logic
}

// Handling errors with assignment
json|error result = someFunction();

// use ! to lift error and 'check' to eliminate
error value.
json name = check result!name!fname;
```

Concurrency

Workers

```
function main(string... args) {
    worker w1 {
        io:println("Hello from worker w1");
    }
    worker w2 {
        io:println("Hello from worker w1");
    }
}
```

Fork/join

```
fork {
    worker w1 {
        int a;
        a -> fork;
    }
    worker w2 {
        string b;
        b -> fork;
    }
} join (all) (map results) {
    int w1Value = check <int> results.w1;
    string w2Value = <string> results.w2;
}
```

Async invocation

```
// Asynchronously invoke slowAdd.
future<int> result = start slowAdd(5, 10);

// Wait on the result
int value = await result;
```

Ballerina CHEATSHEET

Transactions

```
transaction with retries = 2 {
    // do something
    if (someCondition) {
        abort;
    }
    if (anotherCondition) {
        retry;
    }
} onretry {
    // do something before retrying
}
```

Security - Taint Analysis

Passing tainted data to a security sensitive param

```
function main(string... args) {
    string input = args[0];
    // proper data validation / sanitization
    if(check input.matches("[a-zA-Z]+")) {
        // use untaint unary expression
        secure(untaint input);
    }
}

// example security sensitive function
function secure(@sensitive string input) {}
```

Defining security sensitive parameters

```
function f1(@sensitive string input) {}
```

Defining untainted return value

```
function f1(string input) returns @untainted
string {
  // proper data sanitization on input
  return sanitized_input;
}
```

Defining tainted return value

```
function f1() returns @tainted string {
    // return untrusted data
}
```

HTTP Client Invocation

```
import ballerina/io;
import ballerina/http;

endpoint http:Client clientEP {
  url: "http://www.example.com"
};

function main(string... args) {
   http:Response resp = check clientEP-> post("/",
"hello");
   io:println(check resp.getTextPayload());
}
```

Database Client Invocation

```
import ballerina/mysql;
import ballerina/io;

type student record {
  int id;
  string name;
};

endpoint mysql:Client testDB {
   host: "localhost",
   port: 3306,
   name: "testdb",
   username: "test",
   password: "test",
   poolOptions: { maximumPoolSize: 5},
   dbOptions: { useSSL: false }
};

function main(string... args) {
   table<student> tb = check testDB->
   select("SELECT id,name FROM student",
      student);
   foreach (s in tb) {
       io:println("Name:" + s.name);
   }
}
```

Websocket

Websocket Echo server

```
import ballerina/http;

service<http:WebSocketService> echo bind {port:
9090} {
    onText (endpoint caller, string text, boolean
final) {
        _ = caller->pushText(text, final = final);
    }
}
```

Websocket Client

```
import ballerina/http;
import ballerina/io;

function main (string... args) {
    endpoint http:WebSocketClient wsClient {
        url: "wss://echo.websocket.org",
        callbackService: client
    };
    _ = wsClient->pushText("Hello World!");
}

service<http:WebSocketClientService> client {
    onText(endpoint caller, string text, boolean
finalFrame) {
        io:println(text);
    }
}
```

gRPC

gRPC Hello service. File hello.bal:

```
import ballerina/grpc;

service<grpc:Service> Hello bind { port: 9090 } {
   say(endpoint caller, string name) {
        string message = "Hello " + name;
        _ = caller->send(message);
        _ = caller->complete();
   }
}
```

Build the service and generate service proto file.

```
> ballerina build hello.bal

Proto File: grpc/Hello.proto
```

Ballerina CHEATSHEET

gRPC client invocation

```
import ballerina/io;
import ballerina/grpc;

endpoint HelloBlockingClient clientEP {
   url: "http://localhost:9090"
};

function main(string... args) {
   (string, grpc:Headers) payload =
           check clientEP->say("Ballerina");
   string result;
   (result, _) = payload;
   io:println(result);
}
```

Generate client stub code using .proto file generated from the service.

```
> ballerina grpc --input Hello.proto
--output client
```

Package the client code and client stub code.
Build and run client package.

Socket

Socket client

```
import ballerina/io;

function main string... args) {
   io:Socket client = new();
   check client.connect("localhost", 9999);
   io:ByteChannel channel = client.channel;
   string text = "Sample Text\n";
   byte[] content = text.toByteArray("UTF-8");
   int count = check channel.write(content, 0);
   check client.close();
}
```

SocketServer

```
import ballerina/io;

function main(string... args) {
    io:ServerSocket server = new();
    check server.bindAddress(9999);
    while (true) {
        io:Socket s = check server.accept();
        check s.close();
    }
    check server.close();
}
```

File IO

Read or write file content as a string

```
import ballerina/io;

function main(string... args) {
    // Read character stream from file.
    io:ByteChannel fileCh = io:openFile(
        "filePath", io:READ);
    io:CharacterChannel char = untaint new
        io:CharacterChannel(fileCh, "utf-8");
    var read = char.read(10);
    var write = char.write("Hello", 0);
    check char.close();
}
```

Read file content as a CSV stream

```
// Read character stream as a CSV stream.
function readCSV(io:CharacterChannel char) {
    io:CSVChannel csvChannel = untaint new
            io:CSVChannel(char);
    while (csvChannel.hasNext()) {
        var result = csvChannel.getNext();
        // result can be destructured using match
        // operator which will give string[] |
        // error | ().
    }
    check csvChannel.close();
}
```