Ballerina CHEATSHEET

ballerina.io

Basic syntax

File hello.bal:

```
import ballerina/io;

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

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

Hello World service

```
import ballerina/http;

service hello on new http:Listener(9090) {
   resource function sayHello(http:Caller caller,
                              http:Request req) {
      var res = caller->respond("Hello, World!");
   }
}
```

Variables

```
string name = "Ballerina";

// the type is inferred for `var`
var age = true;
```

Functions

```
function foo(int a) returns int {
   return a + 1;
}
var result = foo(4);


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

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

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

Values and Types

Simple Basic Types

Type	Values
`int`	42, -71, 0xFF (64-bit integers)
`float`	1.0 1e-17 1f (64-bit binary floating point)
`decimal`	1.0 1.50d (128-bit decimal floating point)
`boolean`	`true`, `false`
`string`	`"\u[1f600]"` `"Hello world"` (Unicode strings)
`()`	`()` (`nil` represents the absence of any other value), `null` in JSON contexts

Arrays/Tuples

```
// variable-length array
int[] a = [1, 2, 3, 4, 5, 6, 7, 8];
a[999] = 100;

// fixed-length array of predefined length
string[2] b = ["apple", "orange"];

// tuple - a list with members of different types
[int, string, int] tuple = [1, "value", 5];
string value = tuple[1];
```

Record/Map/JSON types

```
type Person record {
   string name;
   int age = 20; // field with a default value
   map<string> address?; // optional field
};

Person p = {name: "John", age: 50};
// update a record field with field-access
p.age = 45;

// map with only string fields
map<string> address = {
    street: "Palm Grove",
    city: "Colombo 03",
    country: "Sri Lanka"
};
// update/access map fields with member access
address["code"] = "011";
// string? is the same as string|()
string? code = address["code"];

// JSON is a built-in union of  (), boolean, int,
// float, decimal, string, json[], and map<json>.
// map<json> is a JSON object.
map<json> info = {
    name: "John",
    "age": 50,
    address: {
        street: "20 Palm Grove",
        city: "Colombo"
    },
    contacts: [123, 789]
};

// access JSON object fields with field-access
json|error j = info.name;
```

Object type

```
type Person object {
   string name;

   // initializer method
   function __init(string name) {
      self.name = name;
   }
   // member method
   function getName() returns string {
      return self.name;
   }
};

Person p1 = new ("John");
Person p2 = new Person("Doe");
string name = p1.getName();
```

Union type

```
// define a union-typed variable
int|error value = getValue();

// type test is used to resolve the runtime type
if value is int {
    int x = value + 1; // value is an int here
} else {
    // value is error here
}

function getValue() returns int|error {
    // returns an int or an error
}
```

Control Structures

Conditional

```
int value = 10;

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


// value switch against a given pattern
string animal = getAnimal();
match animal {
   "Mouse" => { io:println("Mouse"); }
   // match “Dog” or “Canine”
   "Dog"|"Canine" => { io:println("Dog"); }
   // “_” matches to any non-error value
   _ => { io:println("Unknown Animal"); }
}
```

Loops

```
// while loop
int i = 0;
while i < 10 {
    i += 1;
    if i == 5 {
        continue;
    }
    if i == 7 {
        break;
    }
}


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

// incremental integer range from start
// expression (inclusive) to end expression
// (exclusive)
foreach var i in 1 ..< 10 {
    io:println(i);
}
```

Error handling

```
// creating an error
error err = error("errorReason",
                  message = "detailed message");
// two error handling approaches
//  1) return errors
return err;
//  2) panic
panic err;


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

// handling a panic by trapping the panic
json|error result = trap someOtherFunction();
```

Concurrency

Async invocation

```
// asynchronously invoke slowAdd
future<int> result = start slowAdd(5, 10);
// wait on the result
int value = wait result;
```

Workers

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

Fork

```
fork {
    worker w1 returns int {
        10 -> w2;
        int x = <- w2;
        return x * 2;
    }
    worker w2 returns int {
        int y = <- w1;
        20 -> w1;
        return y + 10;
    }
}

record {int w1; int w2;} results = wait {w1, w2};
```

Security - Taint Analysis

Passing tainted data to a security-sensitive param

```
public function main(string arg) {
   string input = arg;
   // proper data validation/sanitization
   if !input.startsWith("exec ") {
       // use untaint unary expression
       secure(<@untainted> input);
   }
}
// example security sensitive function
function secure(@untainted string din) {}
```

Defining a return value as untainted

```
function f1(string s) returns @untainted string {
   // return proper sanitized data on input
}
```

Defining a return value as tainted

```
function f2() returns @tainted string {
   // return untrusted data
}
```

Ballerina CHEATSHEET

ballerina.io

HTTP Client Invocation

```
import ballerina/http;

http:Client cl = new("http://www.example.com");

public function main() returns error? {
    http:Response resp = check cl->post("/",
                                         "hello");
    io:println(resp.getTextPayload());
}
```

Database Client Invocation

```
import ballerinax/java.jdbc;

type Student record {|
    int id;
    string name;
|};

jdbc:Client dbClient = new({
    url: "jdbc:mysql://localhost:3306/testdb"});

public function main() returns error? {
    table<Student> tb = check dbClient->
        select("SELECT id, name FROM student",
Student);
    foreach Student student in tb {
        io:println("Name: ", student.name);
    }
}
```

WebSocket

WebSocket Echo Server

```
import ballerina/http;

service echo on new http:Listener(9090) {
    resource function onText(http:WebSocketCaller
caller, string data, boolean finalFrame) {
        error? result = caller->pushText(data,
finalFrame);
    }
}
```

WebSocket Client

```
import ballerina/http;

public function main() {
    http:WebSocketClient wsClient =
new("ws://echo.websocket.org", {callbackService:
clientService});
    error? res = wsClient->pushText("Hello World!");
}

service clientService =
@http:WebSocketServiceConfig {}
service {
    resource function onText(http:WebSocketClient
caller, string data, boolean finalFrame) {
        io:println(data);
    }
};
```

gRPC

gRPC Hello service. File hello.bal

```
import ballerina/grpc;

service hello on new grpc:Listener(9092) {
    resource function say(grpc:Caller caller, string
name) {
        error? res = caller->send("Hello " + name);
        res = caller->complete();
    }
}
```

Build the service to generate the service proto file.

```
> ballerina build hello.bal
```

Proto file: `grpc/hello.proto`

gRPC Client Invocation

```
import ballerina/grpc;

helloBlockingClient grpcClient =
new("http://localhost:9092");

public function main() returns error? {
    [string, grpc:Headers] [result, _] = check
grpcClient->say("Ballerina");
    io:println(result);
}
```

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

```
> ballerina grpc --input grpc/hello.proto --output
client
```

NATS

Subscriber

```
import ballerina/nats;

nats:Connection connection =
                      new("nats://localhost:4222");
listener nats:Listener subscription =
                      new(connection);

@nats:SubscriptionConfig {
    subject: "demo"
}
service demo on subscription {
    resource function onMessage(nats:Message msg,
string data) {
        log:printInfo("Received message: " + data);
    }

    resource function onError(nats:Message msg,
nats:Error err) {
        log:printError("Error occurred", err);
    }
}
```

Publisher

```
import ballerina/nats;

public function main() {
    nats:Connection connection =
                         new("nats://localhost:4222");
    nats:Producer producer = new(connection);
    nats:Error? result = producer->publish("demo",
                         <@untainted> "message");
}
```

Dockerfile Generation

```
import ballerina/docker;

// generates the Dockerfile for the ballerina
// service and creates the docker image
@docker:Config {}
@http:ServiceConfig {
    basePath: "/helloWorld"
}
service helloWorld on new http:Listener(9090) {
    resource function sayHello(http:Caller caller,
http:Request request) {
        var result = caller->respond("Hi!");
    }
}
```

Kubernetes Artifact Generation

```
import ballerina/kubernetes;
// generates a kubernetes service yaml file
@kubernetes:Service {
    serviceType: "NodePort"
}
// generates a kubernetes yaml file and creates // the
docker image for the ballerina service
@kubernetes:Deployment {
    namespace: "demo-ns",
    replicas: 2,
    image: "helloworld:2.1.0"
}
@kubernetes:ConfigMap {
    configMaps:[{
        mountPath: "/home/ballerina/data",
        data: ["./conf/data.txt"]
    }]
}

@http:ServiceConfig {
    basePath: "/helloWorld"
}
service helloWorld on new http:Listener(9090) {
    resource function sayHello(http:Caller caller,
http:Request request) {
        var result = caller->respond("Hi!");
    }
}
```

Istio Artifact Generation

```
import ballerina/kubernetes;
import ballerina/istio;

@kubernetes:Service {}
// generates istio gateway artifact
@istio:Gateway {}
// generates istio virtual service artifact
@istio:VirtualService {}
listener http:Listener helloEP = new(9090);

@kubernetes:Deployment {}
@http:ServiceConfig {
    basePath: "/helloWorld"
}
service helloWorld on new helloEP {
    resource function sayHello(http:Caller caller,
http:Request request) {
        var result = caller->respond("Hi!");
    }
}
```

Openshift Artifact Generation

```
import ballerina/kubernetes;
import ballerina/openshift;

@kubernetes:Service {}
// generates openshift route artifact
@openshift:Route {
    host: {
        domain: "<MINISHIFT_IP>.nip.io"
    }
}
listener http:Listener helloEP = new(9090);

// generates openshift image stream and build
// config
@kubernetes:Deployment {
    namespace: "bal-oc",
    registry: "<DOCKER_REGISTRY_IP>",
    buildImage: false,
    buildExtension: "openshift"
}
@http:ServiceConfig {
    basePath: "/helloWorld"
}
service helloWorld on new helloEP {
    resource function sayHello(http:Caller caller,
http:Request request) {
        var result = caller->respond("Hi!");
    }
}
```