Ada Programming Language Cheatsheet

A quick reference guide to the Ada programming language, covering syntax, data types, control structures, and advanced features for both beginners and experienced programmers.

Basic Syntax and Data Types

Program Structure

An Ada program consists of a main procedure and optional subprograms (procedures and functions).

```
with Ada.Text_IO;
use Ada.Text_IO;

procedure Hello_World is
begin
   Put_Line ("Hello, World!");
end Hello_World;
```

with clause: Imports packages.
use clause: Makes package contents directly visible.
procedure: Defines the main program.

Data Types

`Integer`	Represents whole numbers. Example: `My_Integer : Integer := 42;`
`Float`	Represents floating-point numbers. Example: `My_Float : Float := 3.14159;`
`Boolean`	Represents truth values (`True` or `False`). Example: `My_Boolean : Boolean := True;`
`Character`	Represents single characters. Example: `My_Character : Character := 'A';`
`String`	Represents a sequence of characters. Example: `My_String : String := "Hello";`

Variable Declaration

Variables are declared with a name, type, and optional initial value.

```
Variable_Name : Data_Type := Initial_Value;
```

Example:

```
Counter : Integer := 0;
Message : String := "Initial Message";
```

Control Structures

Conditional Statements

if statements allow conditional execution of code.

```
if Condition then
   -- Code to execute if condition is true
elsif Another_Condition then
   -- Code to execute if another condition is
true
else
   -- Code to execute if all conditions are
false
end if;
```

Example:

```
if Score >= 60 then
   Put_Line ("Pass");
else
   Put_Line ("Fail");
end if;
```

Looping Statements

`for` loop	Iterates a specific number of times.
	`for i in 1 .. 10 loop`
	`  -- Code to execute`
	`  Put_Line (Integer'Image(i));`
	`end loop;`

`while` loop	Executes as long as a condition is true.
	`Counter := 0;`
	`while Counter < 10 loop`
	`  -- Code to execute`
	`  Counter := Counter + 1;`
	`  Put_Line (Integer'Image(Counter));`
	`end loop;`

`loop` statement	General loop construct, useful with `exit when` .
	`loop`
	`  -- Code to execute`
	`  exit when Condition;`
	`end loop;`

Case Statements

The `case` statement allows selecting one of several code paths based on a value.

```
case Expression is
   when Choice_1 =>
      -- Code to execute
   when Choice_2 =>
      -- Code to execute
   when others =>
      -- Code to execute if no other choice
matches
end case;
```

Example:

```
case Grade is
   when 'A' =>
      Put_Line ("Excellent");
   when 'B' =>
      Put_Line ("Good");
   when others =>
      Put_Line ("Needs Improvement");
end case;
```

Subprograms (Procedures and Functions)

Procedures

Procedures are subprograms that perform actions but do not return a value.

```
procedure Procedure_Name (Parameter_List) is
   -- Declarations
begin
   -- Statements
end Procedure_Name;
```

Example:

```
procedure Greet (Name : String) is
begin
   Put_Line ("Hello, " & Name & "!");
end Greet;
```

Functions

Functions are subprograms that perform calculations and return a value.

```
function Function_Name
(Parameter_List) return
Return_Type is
   -- Declarations
begin
   -- Statements
   return Return_Value;
end Function_Name;
```

Example

```
function Add (A, B : Integer)
return Integer is
begin
   return A + B;
end Add;
```

Parameter Modes

`in`: The default mode. The parameter is read-only.
`out`: The parameter is write-only. Its initial value is not accessible.
`in out`: The parameter can be both read and written.

Overloading

Ada supports overloading of subprograms, allowing multiple subprograms with the same name but different parameter lists.

```
function Add (A, B : Integer) return Integer
is ...
function Add (A, B : Float) return Float is
...
```

Arrays and Records

Arrays

Arrays are collections of elements of the same type, accessed by an index.

```
type Array_Type is array (Index_Range) of
Element_Type;
My_Array : Array_Type;
```

Example:

```
type Int_Array is array (1 .. 5) of Integer;
Numbers : Int_Array := (1, 2, 3, 4, 5);
```

Array Attributes

`Array'First`: Returns the first index of the array.
`Array'Last`: Returns the last index of the array.
`Array'Length`: Returns the number of elements in the array.

Records

Records are collections of named fields, possibly of different types.

```
type Record_Type is record
   Field_1 : Data_Type_1;
   Field_2 : Data_Type_2;
   ...
end record;

My_Record : Record_Type;
```

Example

```
type Employee is record
   ID : Integer;
   Name : String (1 .. 30);
   Salary : Float;
end record;

Emp : Employee;
Emp.ID := 123;
Emp.Name := "John Doe";
Emp.Salary := 50000.0;
```

Accessing Elements

Array elements are accessed using their index within the array.

```
First_Element := Numbers(1);
```

Record fields are accessed using the dot notation.

```
Employee_Name := Emp.Name;
```