Ada Cheat Sheet

Types	
Typedef	type TYPENAME is VALUE;
Predefined types	Integer, Float, Boolean, Character, String,
get size of type (bits)	TYPENAME'size -> example: Integers'size
Enumeration types	Example Boolean -> true, false type Boolean is (true, false);
Integer types	signed: Integer own: type My_Int is range 1..100; 1-maxInteger: Positive
Unsigned / Modular types	0-maxInteger: Natural
Floating Point Types	type byte is mod 2**8;
Fixed Point Types	type ex_values is digits 10 range -1.0..1.0;
	type ordinary_dist is delta 0.001 range 0.0..1.0; -> 2^-10 type decimal_dist is delta 0.01 digits 9 range 0.0..9_999_999.99;

Composite Types	type Own_String is array (1..10) of Integer;
	type String is array(Positive range <>) of Character;

Record / Struct	Ordinary (not extendable through inheritance):
	type Inventory_Item is record
	      UPC_Code : String(1..20);
	end record;
	
	Tagged (extendable through inheritance):
	type Person is tagged record
	   Name : String(1..20);
	end record;
	type Employee is new Person with record
	     Id : Integer;
	end record;

Subtypes	subtype Rainbow is Color range Red .. Blue;
Ranges	
For scalar types	type Rankings is new Integer range 1..10;
Subtypes	> see subtypes
Loops	

	for Num in 1..10 loop
First	….
Last	end loop;
Range	Days’First
	Days’Last
	Voltages’Range == Voltages’First..Voltages’Last
Operators	
Assignment	:=
Equality	=
NonEquality	/=
Modulus	mod
Remainder	rem
AbsoluteValue	abs
Exponentiation	**
Membership	In
Log AND == Bit AND	and (same: or, xor, not)
String Concatination	&
Constructor / Destructor like blocks	
Constructor with function	`type T is tagged record`
	`   F : Integer := init_function;`
	`end record;`
	`function init_function return Integer is`
	`begin`
	`   Put_Line ("Compute");`
	`   return 0;`
	`end init_function;`
	`V1 : T;`
	`V2 : T := (F => 0);`
Advanced using Initialize and Finalize	`type T is new Ada.Finalization.Controlled with record`
	`   F : Integer;`
	`end record;`
	`procedure Initialize (Self : in out T) is`
	`begin`
	`   Put_Line ("Compute");`
	`   Self.F := 0;`
	`end Initialize;`
	`V1 : T;`
	`V2 : T := (F => 0);`
Loops	
Loop	loop
	if condition then
	exit;
	end if;
	end loop;
While	while condition loop
	…
	end loop

for	for var in low_value .. high_value loop … end loop;
Conditions	

If
 	
Switch case
	If condition then
	…
	end if;
	
	case expression is
	    when choice =>
	        ….
	    when choice2 =>
	        ….
	end case;

Subprograms	
Procedure (no return value) Function (always return value)	procedure function_name(in1, in2 : IN OUT Integer) is Temp : Integer := Left; begin Right := Temp; end function_name; IN OUT -> initial value and expected to be written to IN -> Read Only constant OUT -> No initial value but expected to be written to Only IN parameter
Package handling	

define package
 	
use package
	
	package PACKAGENAME is
	end PACKAGENAME;
	
	with PACKAGENAME;
	use PACKAGENAME;

Concurrency protected type task	
Visibility / inheritance	
Generics / Templates	
Useful Building Blocks	
Std. Output Std. Input	Package Ada.Text_IO / Ada.Integer_Text_IO - Put(OUTPUT) -> single character - Integer: Put(VALUE, Width=>1); -> Width: length value - Put_Line(OUTPUT) ->line ** - Get(s) -> reads s.length input to s (ignores new lines)

	- Get_Line(s, len) -> reads len length input to s
File IO	
Create file	- Filevar : FILE_TYPE; Create(Filevar, Out_File, “filename.txt”);
Write single to file	- Put(Filevar, “output text”)

Set output to file	- Set_Output(Filevar);
	Put(“output text”);
	Put_Line(“output line”);
	New_Line(n); -> n = number of new lines
	Set_Output(Standard_Output);

Close file	- Close(Filevar)
Open file	- Open(Filevar, In_File, “filename.txt”)
Read char	- Get(Filevar, c) -> c = input char

Read line	- loop
	    exit when End_Of_File(Filevar);
	    Get(Filevar, c);
	    If End_Of_Line(Filevar) then
	     …
	    else
	      Put(c);
	    end if
	end loop;

Reset position in file	- Reset(Filevar);
Skip line	- Skip_line;