Basic Types

Presented by Quentin Ochem

university.adacore.com

A Few Syntactical Notes

Identifiers

- **Ada identifiers are case insensitive**
  - HELLO = hello = HellO
- **Start with a letter**
- **Ends with a letter or a number**
- **May contain non-consecutive underscores**

Which of the following are legal?

- Something
- My__Id
- _Hello
- A_67_9
- _CONSTANT
- 09_A_2
- YOP_

Identifiers

- **Ada identifiers are case insensitive**
  - HELLO = hello = HellO
- **Start with a letter**
- **Ends with a letter or a number**
- **May contain non-consecutive underscores**

- **Which of the following are legal?**
  - Something
  - ✗ My__Id
  - ✗ _Hello
  - A_67_9
  - ✗ _CONSTANT
  - ✗ 09_A_2
  - ✗ YOP_

Comments

- **Ada provides end of line comments with --**

```
--  This is an Ada comment
```

```
// This is a C++ comment
```

- **There is no block comment (/* */)**

Numbers

- **The underscore is allowed for numbers**
  - 1_000_000 = 1000000
- **Numbers can be expressed with a base (from 2 to 16)**
  - 10#255# = 2#1111_1111# = 8#377# = 16#FF#
- **Float literals must have a dot**
  - With a digit before and after the dot.
  - 1.0 /= 1

Variable Declaration

- **Defined by one (or several) names, followed by :, followed by type reference and possibly an initial value**

```
A : Integer;
B : Integer := 5;
C : constant Integer := 78;
D, E : Integer := F (5);
```

```
int A;
int B = 5;
const int C = 78;
int d = F (5), e = F(5);
```

- **Elaboration is done sequentially**

```
A : Integer := 5;
B : Integer := A;
C : Integer := D; -- COMPILATION ERROR
D : Integer := 0;
```

- **Initialization is called for each variable individually**

```
A, B : Float := Compute_New_Random;
--  This is equivalent to:
A : Float := Compute_New_Random;
B : Float := Compute_New_Random;
```

- **“:=“ on a declaration is an initialization, not an assignment (special properties, mentioned later)**

Simple Scalar Types

Ada Strong Typing

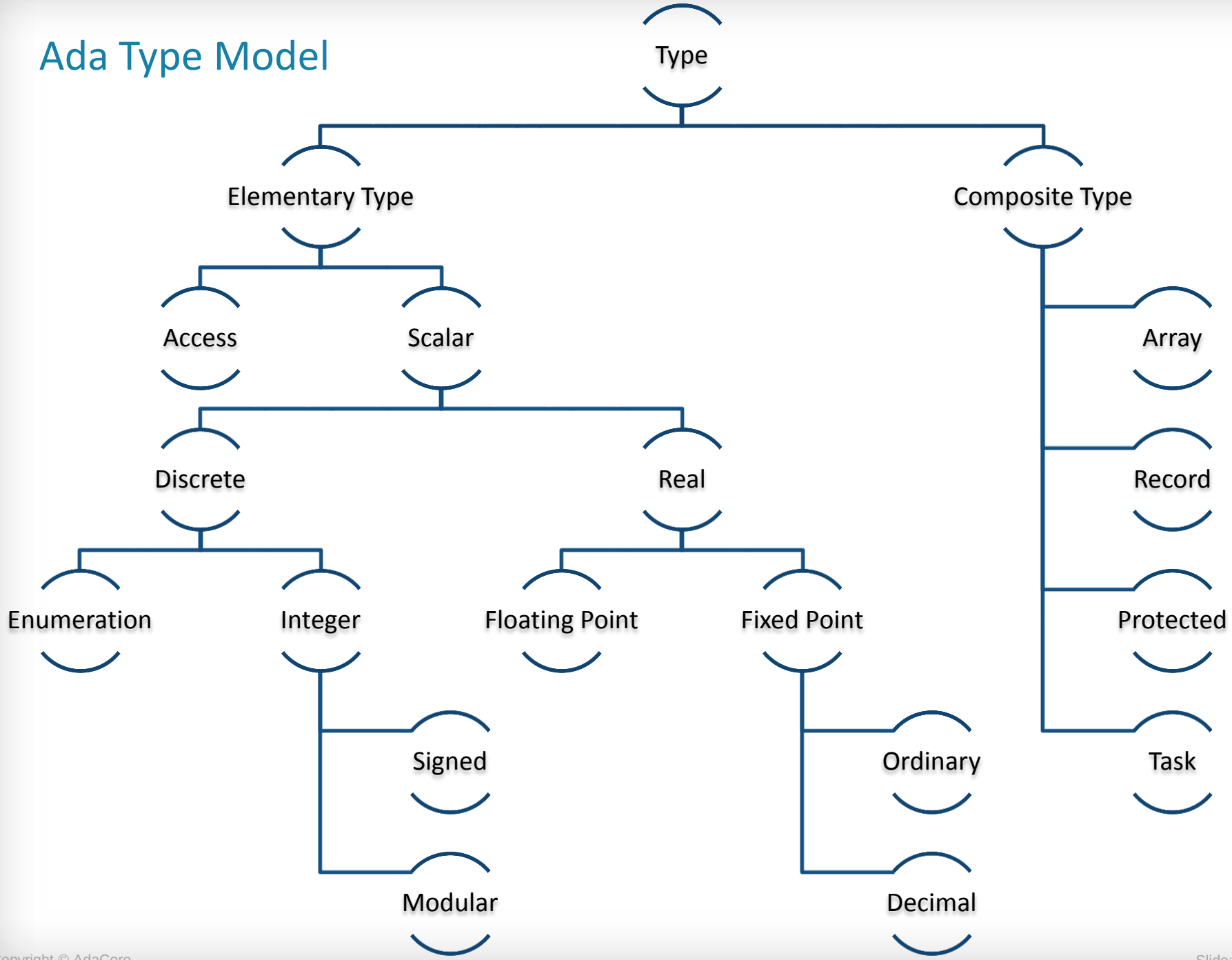
- **Types are at the base of the Ada model**
- **Semantic ≠ Representation**
- **Ada types are named**
- **Associated with properties (ranges, attributes…) and operators**

```
A : Integer := 10 * Integer (0.9);
A : Integer := Integer
   (Float (10) * 0.9);
```

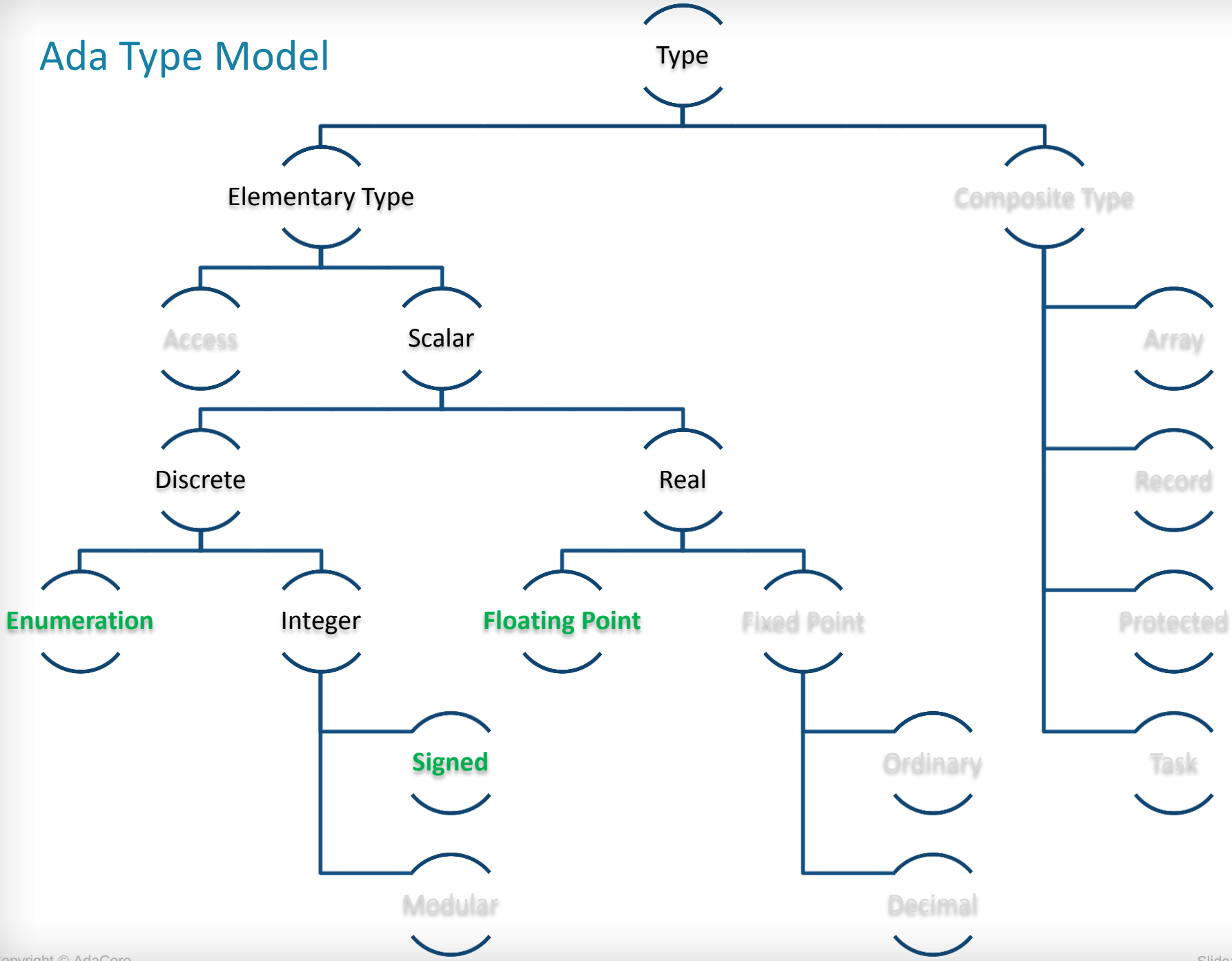
```
int A = 10 * 0.9
```

- **The compiler will warn in case of inconsistencies**

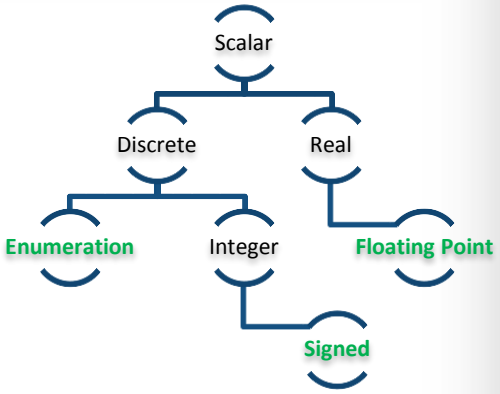
Ada Type Model



Ada Type Model



Scalar, Discrete and Real Types



- **A scalar type is a "single" value**
  - Integer, floating point, enumerations are all scalars
- **Scalar types are divided between "discrete" types (that have a finite number of values) and "continuous" types**
- **Some scalar types are associated with numerical operations**

Standard Types

- **Signed Integers**
  - Short_Integer, Integer, Long_Integer, Long_Long_Integer
- **Enumerations**
  - Character, Boolean
- **Floating Points**
  - Short_Float, Float, Long_Float, Long_Long_Float

Custom Types Declaration

- **Integer types are defined by a range of values**

```
type Score is range 0 .. 20;
type Percentage is range -100 .. 100;
```

- **Enumeration types are defined by a list of values**

```
type Color is (Red, Green, Blue, Yellow, Black);
type Ternary is (True, False, Unknown);
```

- **Float types are defined by a minimal number of significant (decimal) digits**

```
type Distance is digits 10;
type Temperature is digits 5 range -273.15 .. 1_000_000.0;
```

A range will decrease performances of floating point types

Creating a Type from an Existing Type

- **It is possible to create a new type based on an existing type**

```
type Math_Score is new Score;
```

- **The new type can restrict the range of values**

```
type Math_Score is new Score range 0 .. 10;
type Primary_Color is new Color range Red .. Blue;
```

Type Conversion

- **In certain cases, types can be converted from one to the other**
  - They're of the same structure
  - One is the derivation of the other

- **Conversion needs to be explicit**

```
V1 : Float := 0.0;
V2 : Integer := Integer (V1);
```

- **Conversion may introduce a verification**

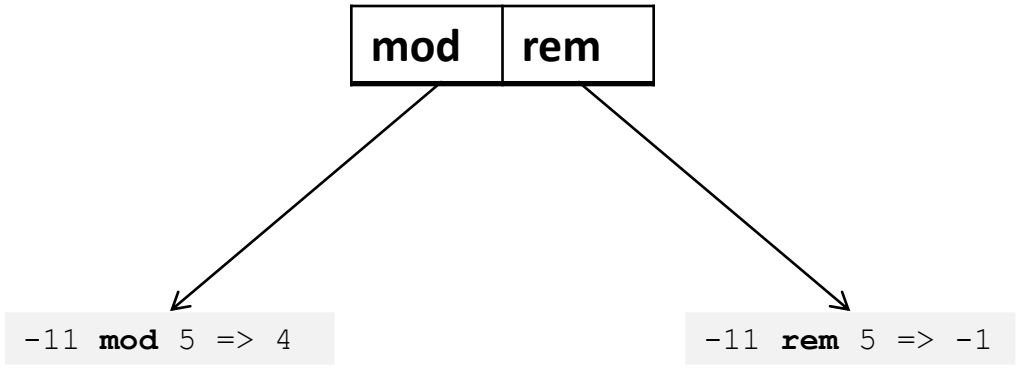
```
type T1 is range 0 .. 10;
type T2 is range 1 .. 10;

V1 : T1 := 0;
V2 : T2 := T2 (V1); -- Run-time error!
```

Ada Operators

- **Common to Signed and Floating Point Types**

=	/=	<	<=	>	>=	+	-	*	/	abs	**

- **Specific to Signed Types**



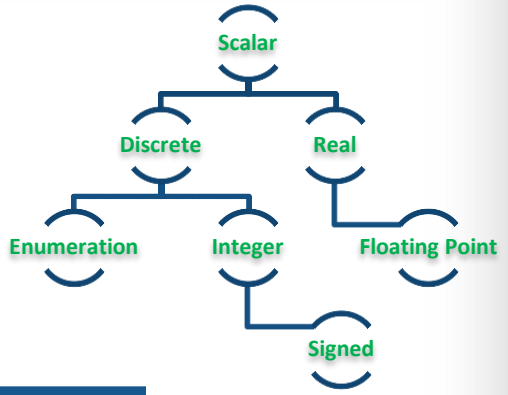
Attributes

- **An attribute is a standard property of an Ada entity**
- **Accessed through '**

```
S : String := Integer'Image (42);
```

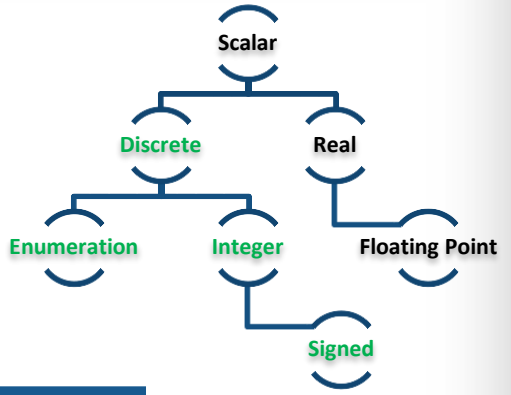
- **Different set of attributes are available depending of the category of the type**

Sample of Attributes Specific to all Scalar



Attribute Name	Documentation
First	Returns the first value of the type
Last	Returns the last value of the type
Image (X)	Converts a value to its corresponding String
Value (X)	Converts a String to its corresponding value
Min (X, Y)	Returns the maximum of two values
Max (X, Y)	Returns the minimum of two values
Pred (X)	Returns the previous value
Succ (X)	Returns the next value
Range	Equivalent of T'First ..T'Last

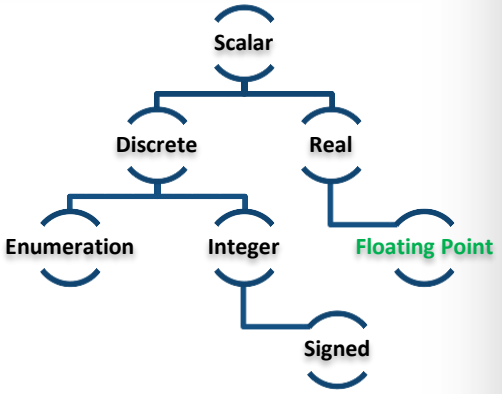
Sample of Attributes Specific to Discrete Types



Attribute Name	Documentation
Pos (X)	Returns the position of the value in the type
Val (X)	Returns a value according to its position

```
V : Character := Character'Val (0);
W : Next_Character := Character'Val (Character'Pos (V) + 1);
```

Sample of Attributes Specific to Floating Point



Attribute Name	Documentation
Ceiling (X)	Returns the smallest integral value after X
Floor (X)	Returns the largest integral value before X
Truncation (X)	Truncates towards 0
Rounding (X)	Rounds to the closest integer
Remainder (X, Y)	Returns the remainder of the Euclidian division

Conversion Float to Integer is using Rounding, not Truncation!

Quiz

Is this correct? (1/10)

YES
(click on the check icon)

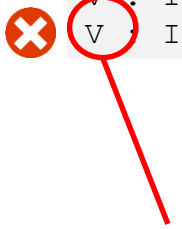
NO
(click on the error location(s))

```
V : Integer := 7;
V : Integer := V + 5;
```

Is this correct? (1/10)

NO

```
V : Integer := 7;
V : Integer := V + 5;
```



Compilation error, V is already declared

Is this correct? (2/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
type N is range -2 ** 256 .. 2 ** 256;
```

Is this correct? (2/10)

NO

```
type N is range -2 ** 256 .. 2 ** 256;
```

This is likely to be to big on most systems today, so the compiler won't be able to represent this and will issue an error.

Is this correct? (3/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
V : Float := 5.0;
```

Is this correct? (3/10)

YES

```
V : Float := 5.0;
```

No Error

Is this correct? (4/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
ClassRoom : constant Natural := 5;
Next_ClassRoom : Natural := classroom + 1;
```

Is this correct? (4/10)

YES

```
ClassRoom : constant Natural := 5;
Next_ClassRoom : Natural := classroom + 1;
```

No Error, Ada is case insensitive

Is this correct? (5/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
type T1 is new Integer range -10 .. 10;

type T2 is new T1 range -100 .. 100;
```

Is this correct? (5/10)

NO

```
type T1 is new Integer range -10 .. 10;

type T2 is new T1 range -100 .. 100;
```

A range cannot be extended, only reduced

Is this correct? (6/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
X : Float := Float'Succ (0.9);
```

Is this correct? (6/10)


```
X : Float := Float'Succ (0.9);
```

Correct. Succ is available for floating point number, and will provide the closest floating point value above the representation of 0.9.

What's the output of this code? (7/10)

```
   F   : Float   := 7.6;
   Div : Integer := 10;
begin
   F := Float (Integer (F) / Div);
   Put_Line (Float'Image (F));
```

What's the output of this code? (7/10)

```
   F   : Float   := 7.6;
   Div : Integer := 10;
begin
   F := Float (Integer (F) / Div);
   Put_Line (Float'Image (F));
```

0.0

Is this correct? (8/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
type T is (A, B, C);

V1 : T := T'Val ("A");
V2 : T := T'Value (2);
```

Is this correct? (8/10)

NO

```
type T is (A, B, C);

V1 : T := T'Val ("A");
V2 : T := T'Value (2);
```

Compilation errors.
T'Val returns a value from a position
T'Value returns a value from a String

Is this correct? (9/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
type T is (A, B, C);

V1 : T := T'Value ("A");
V2 : T := T'Value ("a");
V3 : T := T'Value (" a ");
```

Is this correct? (9/10)

YES

```
type T is (A, B, C);

V1 : T := T'Value ("A");
V2 : T := T'Value ("a");
V3 : T := T'Value (" a ");
```

No Error.
Conversions are case-insentitive, automatically trimmed.

Is this correct? (10/10)

YES
(click on the check icon)

NO
(click on the error location(s))

```
type T is range 1 .. 0;
V : T;
```

Is this correct? (10/10)

YES

```
type T is range 1 .. 0;
V : T;
```

No Error.
T has an empty range, useful in some circumstances.

university.adacore.com