Data types and how to print them

Adapted from materials by Dr. Carrier

Mr Cup - Unsplash

Assigning variables

Syntax:

```
type var_name = value;
```

Assigning variables

Syntax:

```
type var_name = value;
```

How does this differ from Python?

Assigning variables

Syntax:

```
type var_name = value;
```

How does this differ from Python?

C variables are *statically typed*

Assigning variables

Syntax:

```
type var_name = value;
```

List of types:

Assigning variables

Syntax:

```
type var_name = value;
```

List of types:

Integral (integer) types:

Assigning variables

Syntax:

```
type var_name = value;
```

List of types:

- Integral (integer) types:
  - `int` - At least 16 bits

Assigning variables

Syntax:

```
type var_name = value;
```

List of types:

Integral (integer) types:

`int` - At least 16 bits

`long` - At least 32 bits

Assigning variables

Syntax:

```
type var_name = value;
```

List of types:

Integral (integer) types:

- `int` - At least 16 bits
- `long` - At least 32 bits
- `long long` - At least 64 bits

Assigning variables

Syntax:

```
type var_name = value;
```

List of types:

- Integral (integer) types:
    - `int` - At least 16 bits
    - `long` - At least 32 bits
    - `long long` - At least 64 bits
- Floating point types:

Assigning variables

Syntax:

```
type var_name = value;
```

List of types:

Integral (integer) types:

- `int` - At least 16 bits
- `long` - At least 32 bits
- `long long` - At least 64 bits

Floating point types:

- `float` - typically 32 bits

Assigning variables

Syntax:

```
type var_name = value;
```

List of types:

Integral (integer) types:

- `int` - At least 16 bits
- `long` - At least 32 bits
- `long long` - At least 64 bits

Floating point types:

- `float` - typically 32 bits
- `double` - typically 64 bits

Assigning variables

Syntax:

```
type var_name = value;
```

List of types:

Integral (integer) types:

- `int` - At least 16 bits
- `long` - At least 32 bits
- `long long` - At least 64 bits

Floating point types:

- `float` - typically 32 bits
- `double` - typically 64 bits

`char` -

Assigning variables

Syntax:

```
type var_name = value;
```

List of types:

Integral (integer) types:

`int` - At least 16 bits

`long` - At least 32 bits

`long long` - At least 64 bits

Floating point types:

`float` - typically 32 bits

`double` - typically 64 bits

`char` - Character (‘a’, ‘1’, ‘$’, etc)

How to print?

To print just a string:

```
printf(“Hello world!\n”);
```

How to print?

To print just a string:

```
printf(“Hello world!\n”);
```

Syntax of printf:

```
printf(format_str, ...)
```

How to print?

To print just a string:

```
printf(“Hello world!\n”);
```

Syntax of printf:

```
printf(format_str, ...)
```

`format_str` specifies the surrounding text and how to format any variables

How to print?

To print just a string:

```
printf(“Hello world!\n”);
```

Syntax of printf:

```
printf(format_str, ...)
```

`format_str` specifies the surrounding text and how to format any variables

`...` stands for optional arguments, this is where we can pass variables to print

How to print? An example

```
printf("I am %d years old!\n", 5);
```

How to print? An example

```
printf(“I am %d years old!\n”, 5);
```

%d will be replaced by an integer (format specifier)

How to print? An example

```
printf(“I am %d years old!\n”, 5);
```

%d will be replaced by an integer (format specifier)

- We can also use `%i`

How to print? An example

```
printf(“I am %d years old!\n”, 5);
```

`%d` will be replaced by an integer (format specifier)

- We can also use `%i`
- Can specify further formatting:

How to print? An example

```
printf(“I am %d years old!\n”, 5);
```

`%d` will be replaced by an integer (format specifier)

- We can also use `%i`
- Can specify further formatting:
  - `%3d` to specify a minimum width of 3 digits

How to print? An example

```
printf(“I am %d years old!\n”, 5);
```

%d will be replaced by an integer (format specifier)

- We can also use `%i`
- Can specify further formatting:
  - `%3d` to specify a minimum width of 3 digits
    - `%03d` to zero-pad
    - `%-3d` to left justify

How to print? An example

```
printf(“I am %d years old!\n”, 5);
```

%d will be replaced by an integer (format specifier)

- We can also use `%i`
- Can specify further formatting:
  - `%3d` to specify a minimum width of 3 digits
    - `%03d` to zero-pad
    - `%-3d` to left justify

Can also use a variable:

```
int age = 90;
```

```
printf(“I am %d years old!\n”, age);
```

Other format specifiers

`%f` - float or double (floating-point)
%e - exponential notation of float
%g - chooses between normal or exp notation for float

Other format specifiers

`%f` - float or double (floating-point)
`%e` - exponential notation of float
`%g` - chooses between normal or exp notation for float

We can still further customize:

```
printf(“PI = %08.3f”, 3.14);
```

Here, 0 is for zero pad, 8 is for 8 *total* characters (including the decimal point), 3 is for places after decimal point

Other format specifiers

%c -

Other format specifiers

%c - character

Other format specifiers

%c	- character
%s	- string

Signedness

These are different:

```
int x = 5;
unsigned int y = 5;
```

Signedness

These are different:

```
int x = 5;
```

```
unsigned int y = 5;
```

Why would we ever use unsigned ints?

Signedness

These are different:

```
int x = 5;
```

```
unsigned int y = 5;
```

Why would we ever use unsigned ints?

Range!

Unsigned ints can hold numbers twice as large.

Signedness

These are different:

```
int x = 5;
```

```
unsigned int y = 5;
```

Why would we ever use unsigned ints?

Range!
Unsigned ints can hold numbers twice as large.

Technically, ASCII chars are `unsigned chars`

Type conversions (implicit)

C will automatically convert types in certain scenarios:

Type conversions (implicit)

C will automatically convert types in certain scenarios:

- Operating on mismatched types

```
3.0 / 2
```

Type conversions (implicit)

C will automatically convert types in certain scenarios:

- Operating on mismatched types

```
3.0 / 2
```

- Assigning values

```
double x = 5;
int y = 4.0;
```

Type conversions (implicit)

C will automatically convert types in certain scenarios:

- Operating on mismatched types

```
3.0 / 2
```

- Assigning values

```
double x = 5;
int y = 4.0;
```

- Calling functions

```
float_exp(2, 3);
```

Type conversions (explicit)

We can also typecast!

Type conversions (explicit)

We can also typecast!

```
int x = (int) 5.0;
(unsigned int) -1;
```