Getting started

Topics:

1. How to print
2. How to assign variables
3. How to comment
4. Syntax for basic math

How to print

In Julia we usually use `println()` to print

In [28]:
```
println("Welcome to Julia, Manideep !!!")
```

```
Welcome to Julia, Manideep !!!
```

How to assign variables

All we need is a variable name, value, and an equal's sign!
Julia will figure out types for us.

In [29]:
```
my_answer = 42
typeof(my_answer)
```

Out[29]:
```
Int64
```

In [30]:
```
my_pi = 3.14159
typeof(my_pi)
```

Out[30]:
```
Float64
```

In [31]:
```
😺 = "smiley cat!"
typeof(😺)
```

Out[31]:
```
String
```

To type a smiley cat, use tab completion to select the emoji name and then tab again

In [32]:
```
# \:smi + <tab> --> select with down arrow + <enter> ---> <tab> + <
```

After assigning a value to a variable, we can reassign a value of a different type to that variable without any issue.

Out[33]: 1

In [34]:
```
typeof(😺)
```

Out[34]: Int64

Note: Julia allows us to write super generic code, and 😺 is an example of this.

This allows us to write code like

In [37]:
```
😀 = 0
😞 = -1
```

Out[37]: -1

In [38]:
```
😺 + 😞 == 😀
```

Out[38]: true

How to comment

In [39]:
```
# You can leave comments on a single line using the pound/hash key
```

In [40]:
```
#=

For multi-line comments,
use the '#= =#' sequence.

=#
```

Syntax for basic math

In [41]:
```
sum = 3 + 7
```

Out[41]: 10

In [42]:
```
difference = 10 - 3
```

Out[42]: 7

In [43]:
```
product = 20 * 5
```

Out[43]: 100

In [44]:
```
quotient = 100 / 10
```

Out[44]: 10.0

Out[45]: 100

In [46]:
```
modulus = 101 % 2
```

Out[46]: 1

Exercises

1.1

Look up docs for the `convert` function.

In [47]:
```
convert(Int,3.0)
```

Out[47]: 3

1.2

Assign `365` to a variable named `days`. Convert `days` to a float and assign it to variable `days_float`

In [48]:
```
days = 364
days_float = convert(Float32,days)
```

Out[48]: 364.0f0

In [49]:
```
@assert days == 365
@assert days_float == 365.0
```

```
AssertionError: days == 365

Stacktrace:
 [1] top-level scope at In[49]:1
 [2] include_string(::Function, ::Module, ::String, ::String) at .
/loading.jl:1091
```

1.3

See what happens when you execute

```
convert(Int64, "1")
```

and

```
parse(Int64, "1")
```

```
parse(Int64,"1")
```

```
MethodError: Cannot `convert` an object of type String to an objec
t of type Int64
Closest candidates are:
  convert(::Type{T}, !Matched::T) where T<:Number at number.jl:6
  convert(::Type{T}, !Matched::Number) where T<:Number at number.j
l:7
  convert(::Type{T}, !Matched::Ptr) where T<:Integer at pointer.jl
:23
  ...

Stacktrace:
 [1] top-level scope at In[50]:1
 [2] include_string(::Function, ::Module, ::String, ::String) at .
/loading.jl:1091
```