

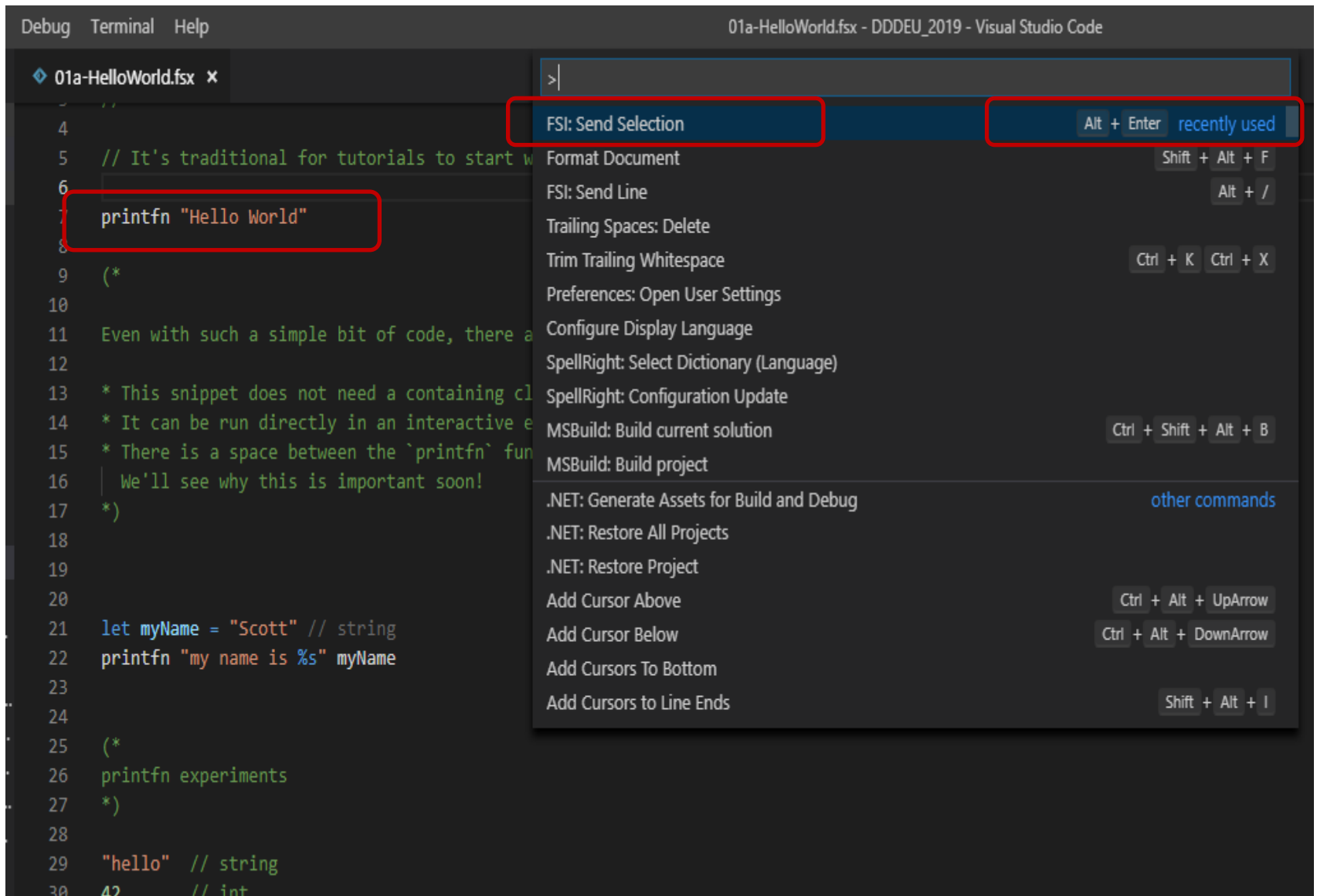
# Getting started with F#

# Using F# with VS Code

- Install VS Code
- Install VS Code extensions:
  - Ionide-fsharp
  - Ionide-paket
- Follow instructions on Ionide-fsharp page:  
<http://ionide.io>

# Test "hello world"

- Open DDDEU\_2019 folder
- Open /src/Day1/01a-HelloWorld.fsx
- To run:
  - Highlight "printfn "Hello World"
  - Ctrl+Shift+P then "FSI: Send Selection"
  - OR just Shift+Enter



```
printfn "hello world"
```


```
let myName = "Scott"  
printfn "my name is %s" myName
```

```
let add x y = x + y  
add 1 2 |> printfn "1 + 2 = %i"
```

a) This snippet does not need a containing class.

b) It can be run directly in an interactive environment

```
printfn "hello world"
```



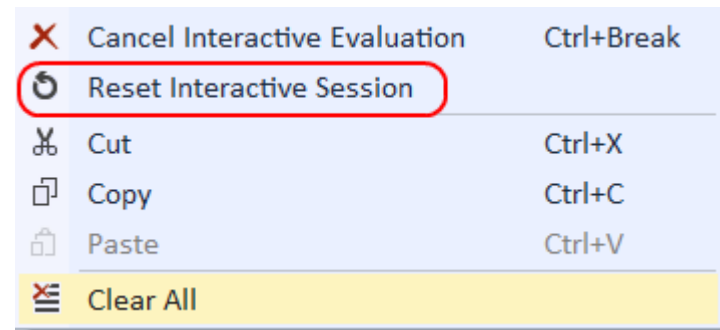
c) There is a space between the 'printfn' function and its parameter, rather than a parenthesis. This will be very important later!

In interactive mode,  
this value is stored globally

```
let myName = "Scott"
```

```
printfn "my name is %s" myName
```

To clear the globals, right  
click and do "reset"



# Documentation

- Syntax cheatsheet:
  - /doc/fsharp-basic-syntax.pdf
  - /src/SyntaxHelp/...
- Help with errors:
  - /doc/TroubleshootingFsharp.pdf



**BASICS**

# Values

String	"hello"
Int	42
Float	3.141
Bool	true, false
List	[1; 2; 3]
Array	[  1; 2; 3  ]
Record	{ name="Scott"; age=27 }

# Declarations

Value	<code>let x= "hello"</code>
Function	<code>let square x = x * x</code>
Type	<code>type Person = {...}</code>

# Things to remember

- Use indentation rather than curly braces
  - No tabs! Spaces only
- Things are not automatically created, you must use "let"
  - "let" is used for values AND functions
- Use spaces for parameter lists
  - No commas
- 0-based collections
- "fun" is a keyword! 😊

comment

```
// F# example
```

Spaces for parameters!

```
let printSquares n =  
    for i in [1..n] do  
        let sq = i*i  
        printfn "%i" sq
```

equals

"let"


Most important difference is invisible: type checking!

# Exercise: Hello World

Evaluate the code in the  
HelloWorld.fsx file

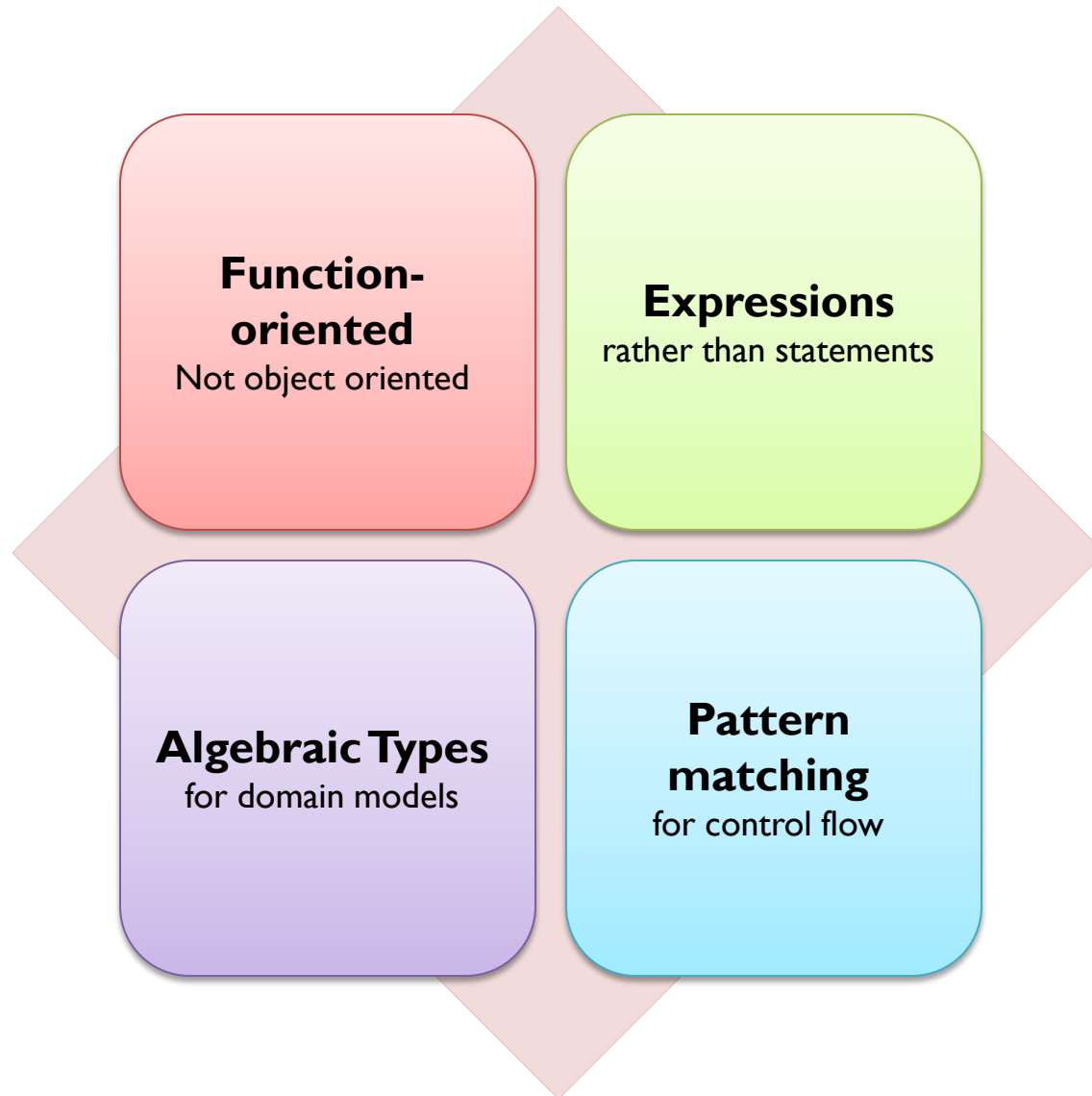
**F# IS DIFFERENT**

# F# vs. languages you're used to

- Differences between F# and other languages
    - Different syntax
    - Type inference
    - Different defaults
    - Different philosophy
  - F# features
    - Functional-first
    - Algebraic type system
    - Interactivity (like a scripting language)
- 
- From least to most important!



# Four things that are very different



# F# has different defaults

- Types must match precisely!
- Immutable by default
  - mutable is special case
- Non-null types/classes by default
  - Nullable is special case
- Structural equality by default
  - reference equality is special case
- Everything must be initialized

**STRICT TYPE CHECKING**

```
1 + 1.5    // what is this?
```



An error!

```
1 + int 1.5    // ok
```

```
float 1 + 1.5    // ok
```

```
1 + "2"    // error
```

```
string 1 + "2"    // ok
```

```
1 + int "2"    // ok
```

**MUTABILITY**

```
let x = 10
```

```
x = 11    // what happens here?
```



*Equality comparison*



```
let x = 10
```

```
x <- 11      // assignment
```

```
let mutable x = 10  
x <- 11      // assignment
```

# Some gotchas

Equality	<code>=</code> (not <code>==</code> )
Inequality	<code>&lt;&gt;</code> (not <code>!=</code> )
Negation	<code>"not"</code> (not <code>!</code> )
Assignment	<code>"let"</code> or <code>&lt;-</code>
Strings	Double quotes (not single)
Parameter separator	space (not comma)
List separator	semicolon (not comma) <code>[ 1; 2; 3 ]</code> <code>{ name="Scott"; age=27 }</code>
Tuples	Comma! <code>(2, 3)</code>
Things to do with types	Colon
Curly braces	For records and similar only

# Exercise: Hello World (2)

Evaluate the rest of the code in the  
HelloWorld.fsx file

# **TYPE INFERENCE**

# Type inference

```
let doSomething f x =  
  let y = f (x + 1)  
  "hello" + y
```



Two parameters: f & x

# Type inference

```
let doSomething f x =  
  let y = f (x + 1)  
  "hello" + y
```

*x must be an int*



*y must be a string*



# Type inference

```
let doSomething f x =  
  let y = f (x + 1)  
  "hello" + y
```

*f must be a 'int -> string' function*

Inferred type of doSomething :

```
f:(int -> string) -> x:int -> string
```

*type of 'f'*

*type of 'x'*

*return type*



# Exercise: Type signatures