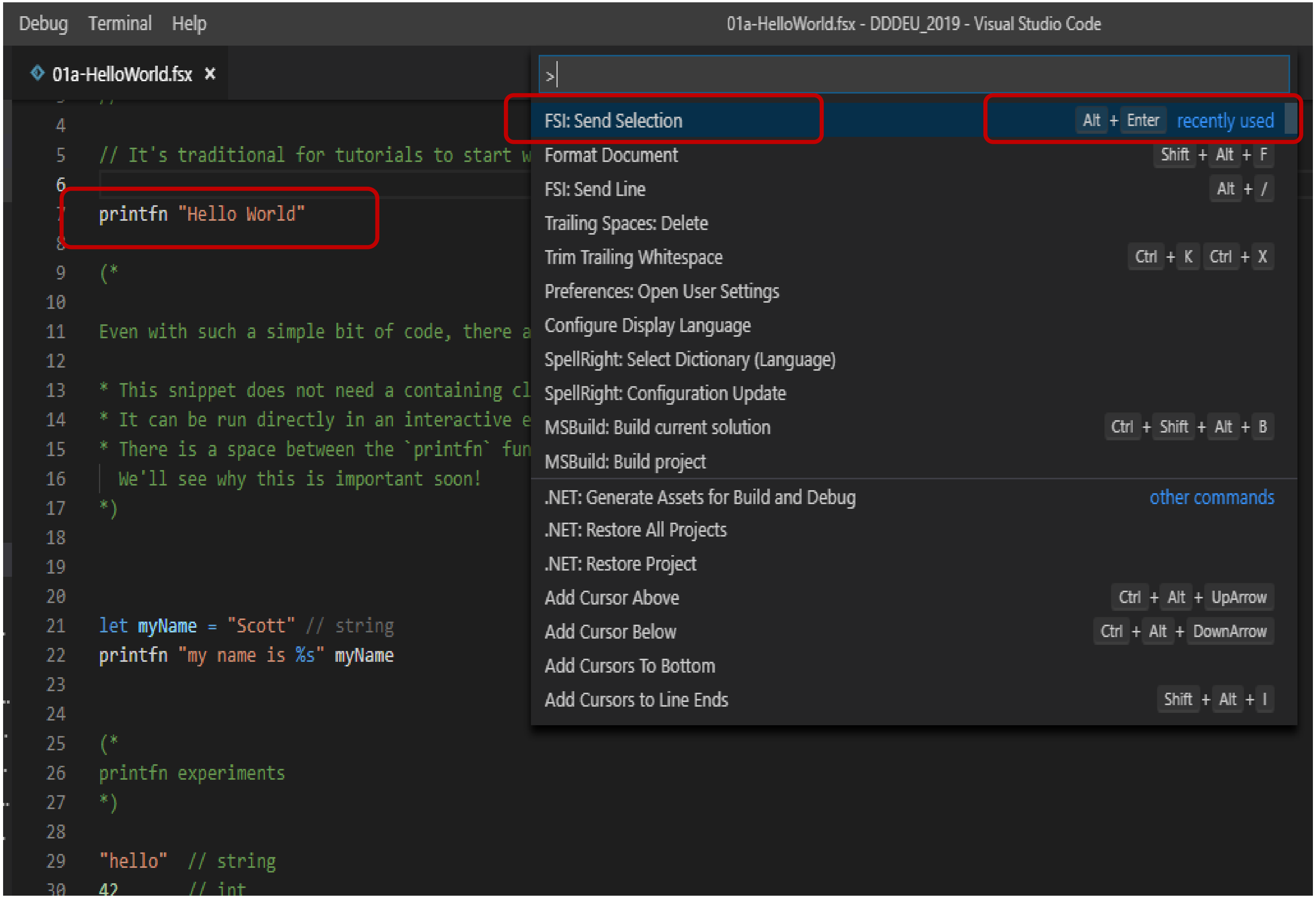
Getting started with F#

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



Debug
Terminal
Help
01a-HelloWorld.fsx - DDDEU_2019 - Visual Studio Code
01a-HelloWorld.fsx ×
>
FSI: Send Selection   Alt + Enter   recently used
Format Document   Shift + Alt + F
FSI: Send Line   Alt + /
Trailing Spaces: Delete
Trim Trailing Whitespace   Ctrl + K Ctrl + X
Preferences: Open User Settings
Configure Display Language
SpellRight: Select Dictionary (Language)
SpellRight: Configuration Update
MSBuild: Build current solution   Ctrl + Shift + Alt + B
MSBuild: Build project
.NET: Generate Assets for Build and Debug   other commands
.NET: Restore All Projects
.NET: Restore Project
Add Cursor Above   Ctrl + Alt + UpArrow
Add Cursor Below   Ctrl + Alt + DownArrow
Add Cursors To Bottom
Add Cursors to Line Ends   Shift + Alt + I

```
//

// It's traditional for tutorials to start w

printfn "Hello World"

(*

Even with such a simple bit of code, there a

* This snippet does not need a containing cl
* It can be run directly in an interactive e
* There is a space between the `printfn` fun
  We'll see why this is important soon!
*)



let myName = "Scott" // string
printfn "my name is %s" myName


(*
printfn experiments
*)

"hello"  // string
42       // int
```

```
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.

```
printfn "hello world"
```

b) It can be run directly in an interactive environment

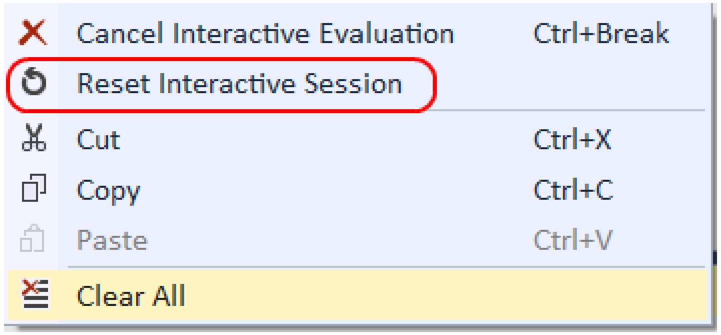
c) There is a space between the 'printfn' function and its parameter, rather than a parenthesis. This will be very important later!

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"


Cancel Interactive Evaluation Ctrl+Break
Reset Interactive Session
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Clear All

Documentation

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

BASICS

Values

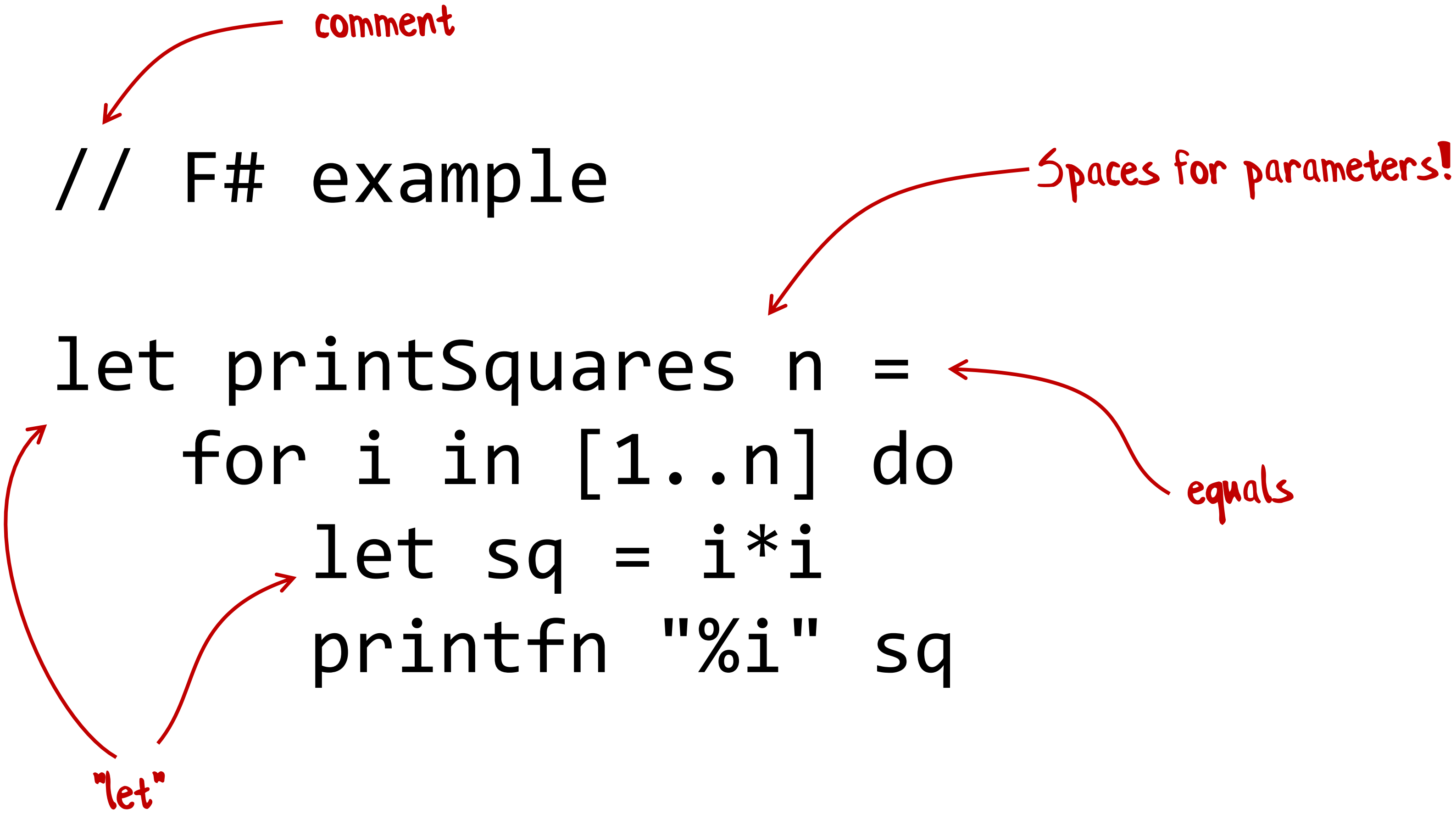
String	"hello"
Int	42
Float	3.141
Bool	true, false
List	[1; 2; 3]
Array	[| 1; 2; 3 |]
Record	{ name="Scott"; age=27}

Declarations

Value	let x= "hello"
Function	let square x = x * x
Type	type Person = {…}

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! ☺



```
// F# example

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

comment

Spaces for parameters!

equals

"let"

Most important difference is invisible: type checking!

Exercise: Hello World

Evaluate the code in the HelloWorld.fsx file

F# IS DIFFERENT

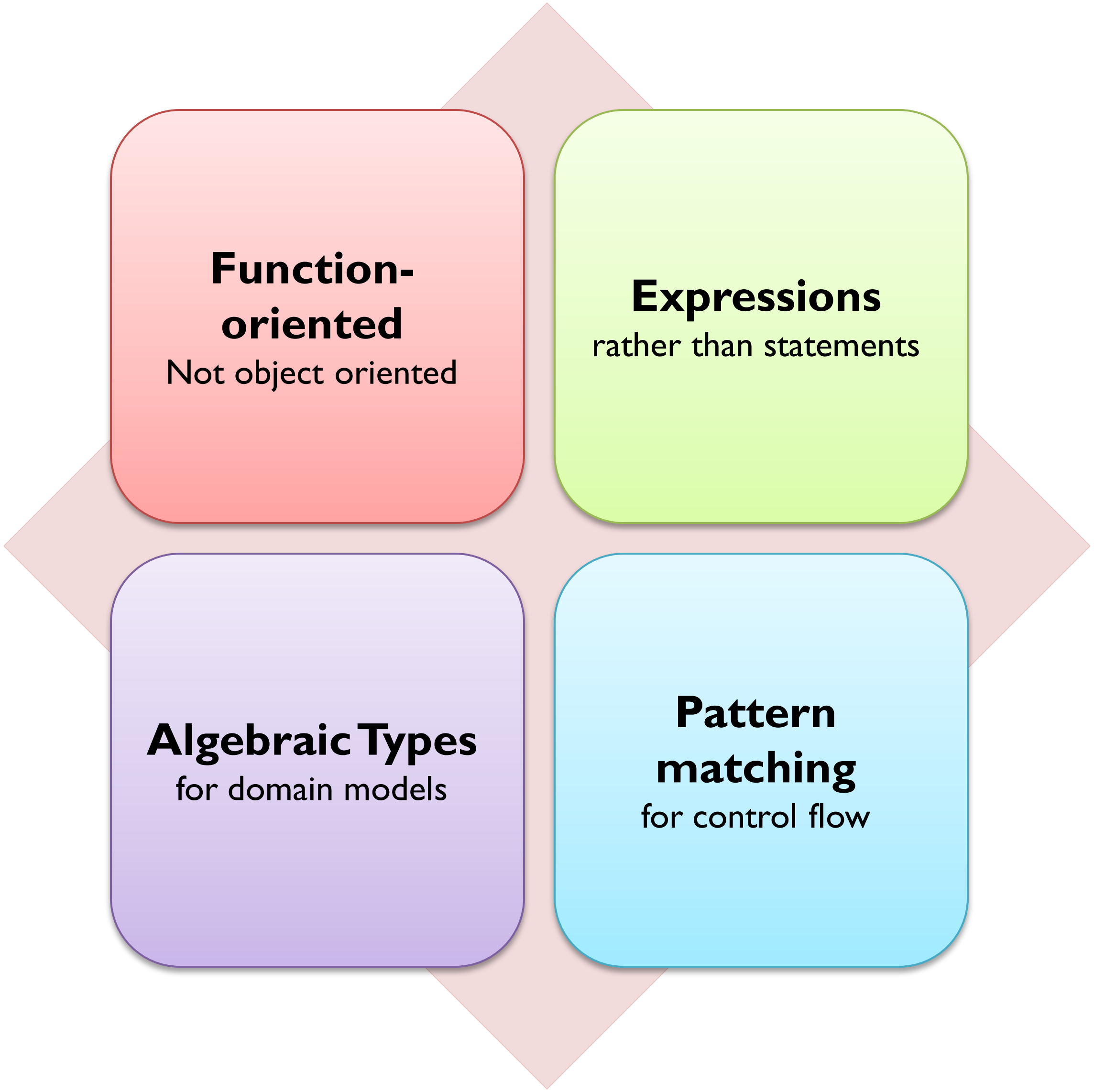
F# vs. languages you're used to

- Differences between F# and other languages
  - Different syntax
  - Type inference
  - Different defaults
  - Different philosophy

From least to most important!

- F# features
  - Functional-first
  - Algebraic type system
  - Interactivity (like a scripting language)

Four things that are very different



Function-oriented
Not object oriented

Expressions
rather than statements

Algebraic Types
for domain models

Pattern matching
for control flow

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	= (not ==)
Inequality	<> (not !=)
Negation	"not" (not !)
Assignment	"let" or <-
Strings	Double quotes (not single)
Parameter separator	space (not comma)

List separator	semicolon (not comma)
	`[ 1; 2; 3 ]`
	`{ name="Scott"; age=27}`

Tuples	Comma! `(2,3)`
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