Defining variables

Constant

```
let consta ntVar = " Hello, World! "
```

Implicit

```
var implic itVar = " Hello, World! "
```

Explicit

```
var explic itVar: String = " - Hello, World! "
```

Loop types

Range

```
for x in 1...5
```

Iterate through array

```
for item in [1, 2, 4, 5]
```

Iterate through dictionary

```
for (key, value) in ["on e":1, " two ":2, ...]
```

Switch methods

```
switch ["1", " 2", " 3", 4, 5, 6, (1, 100)] {
       case " 1":
                pri nt( " 1")
       case " 2", " 3":
                pri nt( "2 or 3")
                pri nt( "No fallth rou gh")
       case 4...5:
                pri nt( " Int - erval matchi ng")
       case (1...100, 1...100):
                pri nt( " Tuples work the same.")
        def ault:
                pri nt( " 6")
                pri nt( " Req - uired in switch s")
                pri nt( "Must be last")
}
```

TODO: _, value binding, fallth - rough, where

Function signature anatomy

Argument Label	*(optional)* Used when calling the function
Parameter Name	Used in the implementation of the function
Return Type	*(optional)* Can be a tuple for multiple return values
throws	*(optional)* Causes errors to propagate up.

```
func FunctionName(ArgumentLabel ParameterName: String) throws -> <ReturnType> {
    // Code
}
```

Event handling

No language constructs available.

Error handling

Representing errors

```
enum ErrorEnum: Error {
       case ErrorTypeA
       case ErrorT ypeB: String
}
```

Propag ating Errors Using Throwing Functions

```
func canThr owE rrors() throws -> String
```

Handling Errors Using Do-Catch

```
do {
       try results = ThrowT - ypeB()
} catch ErrorE num.Er ror Typ - eB(let errorS tring) {
       // Handle error
}
```

Converting Errors to Optional Values

```
let x = try? someTh row ing - Fun ction()
```

Disabling Error Propag ation

```
let x = try! someTh row ing - Fun ction()
```


Sponsored by **Readable.com**
Measure your website readability!
https://readable.com