
The Next Wave
Features in the new wave of languages

The Next Wave

New programming languages come in waves, and there's a bunch of new ones on their way.

Learning about their features can help us prepare, and also help us think in new ways.

The Next Wave

• Swift	Apple	Scala
• Dart	Google	Go
• Rust	Mozilla	Clojure
• Hack	Facebook	C#
• Kotlin	JetBrains	TypeScript
• Ceylon	Red Hat	

The Next Wave

What do they all have in common?

- ✓ Type Safety
- ✓ Type Inference
- ✓ Immutable Data
- ✓ Generics
- ✓ Tuples, Records
- ✓ Tagged Unions
- ✓ Pattern Matching
- ✓ Blocks / Lambdas

Type Safety

Make the compiler do the work

Type Safety

The collection types we know and love:

- NS*Array — Ordered collection of objects
- NS*Dictionary — Key/value pairs of objects

```
@[1, 2, 3]
@[a: 1, b: 2]
```

NSSet

Type Safety

NSArray is **heterogeneous**

```
[arrayOfStrings addObject:@42];   // Sure, no problem!

int n = arrayOfStrings[0].length; // BOOM!!!
```

*** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: '-[NSNumber length]: unrecognized selector sent to instance 0xc302'

Tuples

Values, values, values

Tuples

How do you return multiple values
...in a type safe language?

```
return [42, ”foo”, view] ?
```

Nope. Type safe arrays are **homogenious**
...you can’t combine types

Tuples

How do you return multiple values
...in a type safe language?

```
return [id: 42, name: “foo”] ?
```

Nope. Dictionaries are also **homogenious**
...you can’t use any old values

Tuples

How do you return multiple values
...in a type safe language?

```
return (42, “foo”, view) ?
```

Sure! The type above is called a **tuple**
It’s like a literal, anonymous C struct.

Tuples

What other uses are there?

```
(x, y) = (y, x)          // Swapping values

foo ((42, “foo”))        // Passing multiple values

(x, y) = point           // Extracting multiple values
```

Tuples

You can also create an array of tuples…

```
[ (42, “foo”, view),
  (100, “bar”, view) ]
```

...as long as the tuples all have the same structure, it’s all perfectly type safe!

Tuples

So now we have one more literal type

```
Array<int>                  [1, 2, 3, 4, 5]
Dictionary<string, int>     [“joe”:10, “sue”:20]
Tuple<int, string>          (42, “foo”, view)
```

They are also very useful when used with **pattern matching**

Type Inference

Make the compiler do **all** the work

Type Inference

```
NSMutableArray *array =
  [NSMutableArray arrayWithArray:@[@1, @2, @3]];
```

Wow! That’s a lot of typing and is hard to read!

```
var array = [1, 2, 3]
```

Now we’re talking!

Type Inference

Types can be inferred by the compiler, if it's smart enough.

You can still write out types to be clear and write self documenting code.

“It is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail.”

~ Abraham Maslow, 1966

Interface vs implementation

- NSArray — Ordered collection
- NSDictionary — Key/value pairs

Contiguous memory? Virtual memory?

Binary tree? Red-black tree? Magic?

Tuples

You can use a dictionary as an array:

```
dict = [0: “foo”, 1: “bar”]
str = a[0]
```

Generally, not a good use of memory or cpu.

...but has a specific use case: sparse arrays

Patterns

Assignment Pattern

x = 10;

(x, y) = (10, 20)

Tuples

Array	[1, 2, 2, 3]	Contiguous
Dictionary	[a: 1, b: 2]	Key/value pairs
Set	[1, 2, ~~2~~, 3]	Dict of keys only
List	[1, [1, Null]]	Two branch tree
Record	{a::int, b::string}	

let and var

```
let immutableArray = [1, 2, 3]
var mutableArray = [1, 2, 3]

NSArray *immutableArray = @[@1, @2, @3];  // Boxing
NSMutableArray *mutableArray =
  [[NSMutableArray alloc] initWithArray:@[@1, @2, @3];

[mutableArray addObject:@”foo”];
```

Death to null!!!

Long live Nullable

```
Optional<UIView> parent;
```