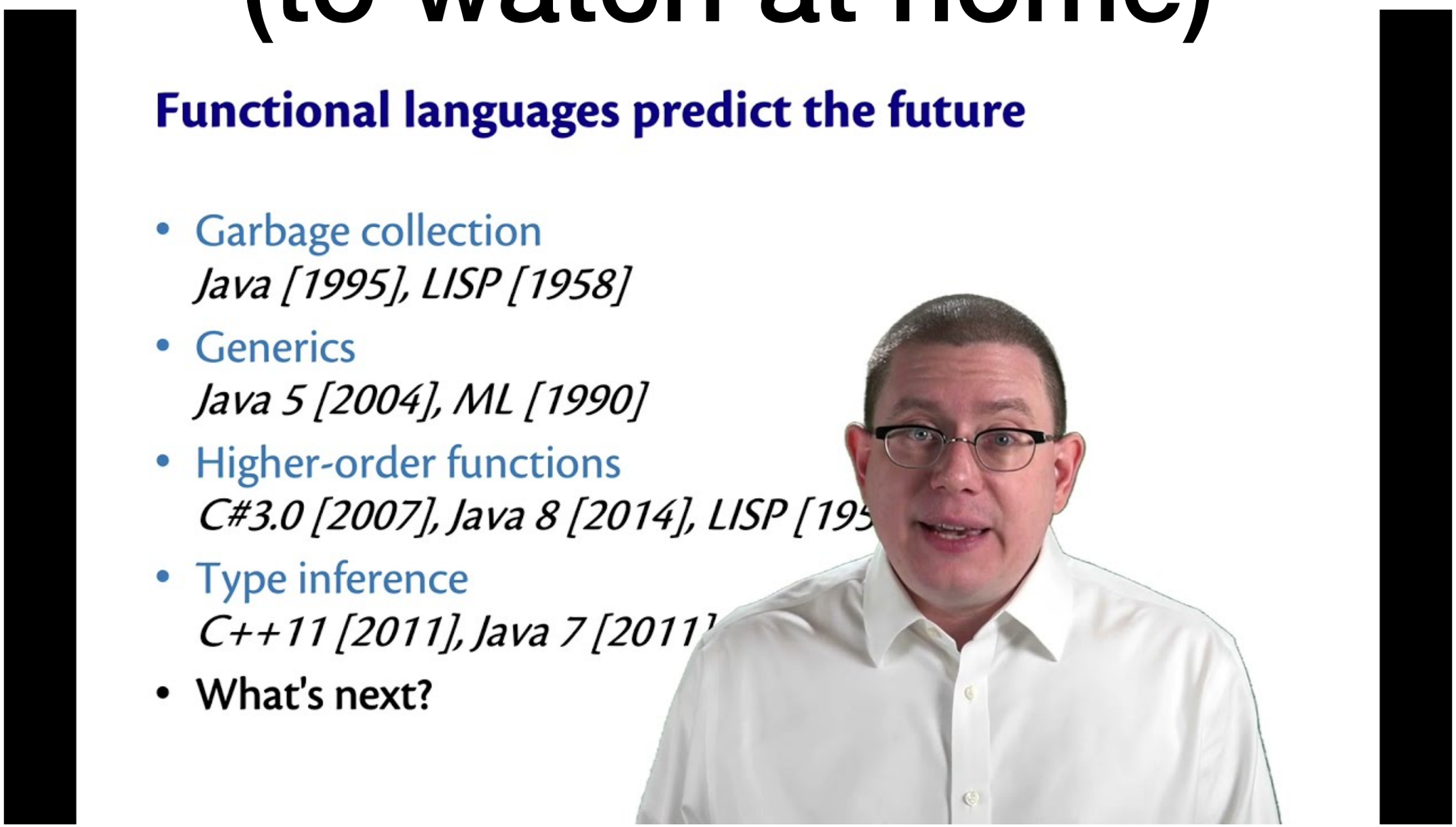
CSE216
Programming Abstraction

State University of New York, Korea

Why Harvard, MIT, Stanford, Cambridge all teach functional programming

https://youtu.be/6APBx0WsgeQ

Another good explanation (to watch at home)



https://youtu.be/SKr3ItChPSI

Install Ocaml

- Official guide: https://ocaml.org/docs/up-and-running
- SBU Guide: https://sites.google.com/cs.stonybrook.edu/cse216/lectures?authuser=0 (may need SBU credentials)
- Once installed, get into the toplevel by running “ocaml”. In the toplevel, run: *print_endline “hello”;;*

```
OCaml version 4.14.0
Enter #help;; for help.

# print_endline "hello";;
hello
- : unit = ()
```

- If nothing works, use TryOcaml for now: https://try.ocamlpro.com/.

Toplevel Demo

```
# 42;;

- : int = 42

# let f x y = x + y;;

val f : int -> int -> int = <fun>

# f 3 ;;

- : int -> int = <fun>

# f 3 4 ;;

- : int = 7

# #use "hello.ml";;

hello world!

- : unit = ()
```

Ocaml Basics

Ocaml program = Definitions + Expressions

```
let x = 1 + 2;;
print_int x;;
let y = x * x;;
print_int y;;
```

Definition

Expression

Definition

Expression

Let-binding is a definition

```
# let x = 5 ;;
val x : int = 5

# let f x y = x + y ;;
val f : int -> int -> int = <fun>
```

```
# let f = (fun x y -> x+y);;
val f : int -> int -> int = <fun>

# let a = f 2 3;;
val a : int = 5

# let g = f 2;;
val g : int -> int = <fun>
```

A note on “let x = e”

Java	OCaml
`final int x = 42;`	`let x = 42`

- **let x = e** defines a global variable
- Differences with regard to Java variables:
- 1 necessarily initialized
- 2 type not declared but inferred
- 3 cannot be modified in-place

A note on functions

```
# let f = (fun x y -> x+y);;
val f : int -> int -> int = <fun>
# let f x y = x + y;;
val f : int -> int -> int = <fun>
```

```
# f 2 3;;
- : int = 5
# f 2 ;;
- : int -> int = <fun>
```

- Similar to λ x. λ y. x + y

Everything other than the definition is an expression

Constants and their operations are expressions

```
# 42;;
- : int = 42
# 42.8;;
- : float = 42.8
# "hello";;
- : string = "hello"
# true;;
- : bool = true
# 'a';;
- : char = 'a'
# 1+1;;
- : int = 2
# 1+1=2;;
- : bool = true
```

Functions and their operations are expressions

```
# fun x y -> x +. y;;

- : float -> float -> float = <fun>

# (fun x y -> x +. y) 2.0 3.7;;

- : float = 5.7

# (fun x y -> x +. y) 2.0;;

- : float -> float = <fun>
```

Exercise: What happens if we do (fun x y -> x +. y) 2 3.7;;

Expression let … in …

let x = e1 in e2 is an expression

its type and value are those of e2,
in an environment where x has the type and value of e1

example

```
let x = 1 in (let y = 2 in x + y) * (let z = 3 in x * z)
```

Semantics of “let…in” is substitution in lambda calculus

Exercise: What is the value and type of above?

A note on the let … in expression

in C or Java, the scope of a local variable extends to the bloc:

```
{
  int x = 1;
  ...
}
```

in OCaml, a local variable is introduced with `let in`:

```
let x = 10 in x * x
```

as for a global variable:

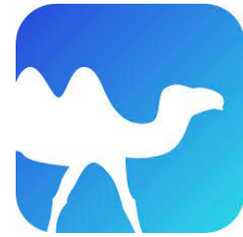
- necessarily initialized
- type inferred
- immutable
- but scope limited to the expression following `in`

Expression
if c then e1 else e2

```
# let f x y = x + y;;

val f : int -> int -> int = <fun>

# if f 4 5 == 9 then "hello" else "world";;

- : string = “hello"
```

e1 and e2 must be of the same type

- Evaluate the boolean condition c.
- If c is true, evaluate the expression e1 and return its value as the result of the if expression.
- If c is false, evaluate the expression e2 and return its value as the result of the if expression.



Unit type

Expressions with no meaningful values have type **unit**.

Example: print_string “okay”

The type has a single value, written **()**

it is the type given to the `else` branch when it is omitted

```
# let x = 5;;
val x : int = 5
# if x > 0 then print_string "okay";;
okay- : unit = ()
# if x > 0 then 100;;
Error: This expression has type int but an expression was expected of type
         unit
       because it is in the result of a conditional with no else branch
# ■
```

Expression e1;e2

- The first expressions e1 needs to be of type unit
- The evaluation result of the whole is that of e2

```
# print_string "hello"; 3 ;;
hello
- : int = 3
# 4①; 3 ;;
- : int = 3
① Warning : this expression should have type unit.
```

Exercises

- Evaluate let x = 3 in let y = x + 2 in y*y

- Evaluate let x= 3 in (let y= x+1 in y) * (let z = x * x in z)

Exercises

- For each of these expressions, what is its value and type. Write the value of a function by “<fun>”:
  - let x = 3 in x * x
  - print_string “hello”
  - print_string
  - let f x y = x + y in f 5 6
  - let x= 5 in (if x > 0 then "pos" else “neg”)
  - let f x y = x + y in f 5

Summary so far

- Compiling/Interpreting an Ocaml program
- Ocaml program = Definitions + Expressions
  - let x = 3 is a definition
  - Everything else is an expression, thus has a value and type
  - let x = 3 in x+5 is an expression of value 8, type int
  - if x then 3 else 5 is also an expression, of type int

Exercise 1: Hello World

- Task: Write a program that prints "Hello, World!" followed by a new line.

Exercise 2: Debugging Type Error

- Try combining x and y using the ^ operator. What error do you see? Choose your own x and y

- Debug and make things right

Exercise 3: Basic Arithmetic

- Do research on the Taylor expansion of e^x around x = 0

- Write a function **taylor** that computes e^x using the first three items of the Taylor expansion

- Calculate e^x at x = 0.1 with the function taylor. Expected result is 1.105

Exercise 4: Functions with multiple arguments

- Important note: Define a function with “let function_name function_parameters = …”

- Write a function **sum** that takes two integers as arguments and returns their sum.

- Write a function average that takes two float values and returns their average.

-

Exercise 5: Basic Recursion

- Important note: Define a recursive function with “let rec function_name function_parameters = …”

- Write a recursive function **factorial** to calculate the factorial of a number.

- Test Input: 5

- Expected Output: 120

Exercise 6: Conditional Statements

- Write a function **maximum** that takes three integers and returns the largest of them.