PROGRAMMING LANGUAGE FAMILIES

Procedural, Object Oriente
Logic, and Functional Langu

PLAN FOR TODAY

History of Programming Language Families

Procedural Languages

Object Oriented Languages

Logical Languages

Functional Languages

HISTORY

Beenish

EARLY BEGINNINGS

Beginnings of automatic computation - Charles Baggage's Difference

First electronic computation engine - ENIAC

Conditional Control Transfer - John Von Neumann

First Compiler – A-o by Dr. Grace Murray

HISTORY OF PROGRAMMING LANGUAGE FAMILIES

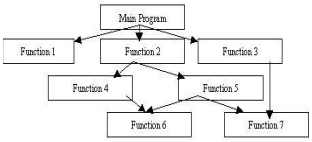
- FORTRAN – originally imperative language, evolved into a procedural language
- Logical programming with languages such as Prolog
- Hope – First functional programming language
- Object oriented began in 1960s
- Considered paradigm that absorbs all other paradigms

Year	Language
1952-53	SHORTCODE, Speedcoding, Laning & Zieler
1954	FORTRAN
1955	FLOW-MATIC
1957	COMIT APT
1958	ALGOL 58 (IAL)
1959-60	COBOL JOVIAL LISP
1960	ALGOL 60
1961	GPSS, SIMSCRIPT
1962	SNOBOL
1964	JOSS BASIC PL/I FORMAC
1966	APL\360
1967	SIMULA 67
1968	ALGOL 68
1970	Pascal
1975	C PROLOG
1976	Ada
1980	Smalltalk-80

PROCEDURAL

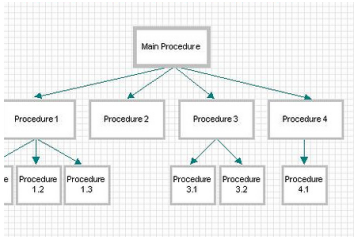
Nick

Procedural Programming Languages



- Traditional, widely used paradigm
- C, PASCAL, C++, Java
- Defined set of instructions
- "main function" calls other functions/ procedures
- Modularity, Scoping

PROCEDURAL PROGRAMMING: MODULARITY



- Procedures act independently
- Easy to group procedures
- Modules can call use other modules
- Intuitive organization of program

PROCEDURAL PROGRAMMING: SCOPING

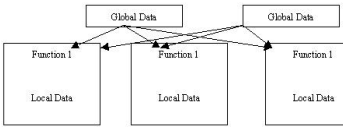
Required data given on procedure call

Can be bypassed by using global data

Data hidden from other procedures

C example:

```
int mult (int x, int y)
{
    return x * y;
}
```



ADVANTAGES AND DISADVANTAGES

ros:

Simple and easy to learn

Logical ordering easy to understand

Can solve most problems

Cons:

- Harder to solve complex problems compared to OOL
- Solutions often require increased amount of complicated code compared to OOL

OBJECT ORIENTED

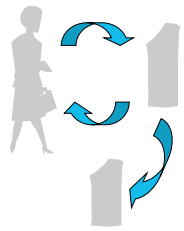
Jobelle

OBJECT ORIENTED PROGRAMMING LANGUAGES

- programs are organized as cooperative collections of objects
- techniques and structure that a programmer use for object orientation

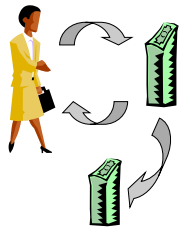
PROCEDURAL VS. OBJECT-ORIENTE

'ACTION', 'PROCEDURE', 'METHOD'



Withdraw, deposit, transfer

'WHAT TO DO', NOT ON 'HOW TO DO'



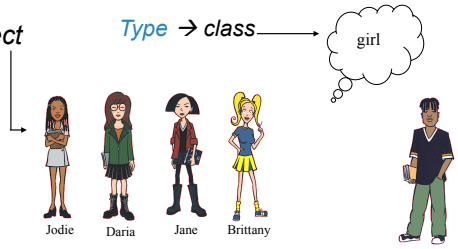
Customer, money, account

OBJECTS AND CLASSES

Classes reflect concepts, objects reflect instances that embody those concepts.

Variable → *object*

Type → *class*



TECHNIQUES

Data abstraction → represents essential features without including background details and explanati

Encapsulation → hiding an object's data

Inheritance → to inherit the variables and methods of an existing class

Polymorphism → create objects of different types

```
class Sports {
        int score;
        int numOfPlayers;
        //constructor
        Sports() {}
        Sports(int x, int y) {
                score = x;
                numOfPlayers = y;
        }
}
class Basketball extends Sports {
        int equipment;
        Basketball(int x, int y, int equip) {
                super(x,y,score);
                //remaining initialization
                equipment = equip;
        }
        //method to shoot
        void shoot() {}
}
```

Inheritance

```
class Fruit {
        public void show() {
                System.out.println("Fruit");
        }
}
class Banana extends Fruit {
        @Override
        public void show() {
                System.out.println("Banana");
        }
        public void makeBananaTree() {
                System.out.println("Making a tree'
        }
}
public class Application {
        public static void main(String[] args) {
                Fruit banana = new Banana();
                banana.show();
                banana.makeBananaTree();
        }
}
```

Polymorphism

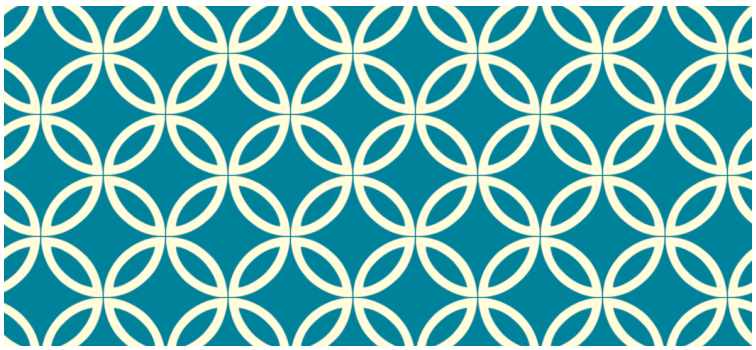
ADVANTAGES AND DISADVANTAGES

shorter development times

easier code sharing

flexibility

- learning and getting used to how OOL works may not be as easy as it seems
- more time to understand and implement object-oriented code
- Not all software engineering problems can be solved or made easier using OOL.



LOGIC | Yunus

Logic Programming Languages

```
ild(trude, sally).

ild(tom, sally).
ild(tom, erica).
ild(mike, tom).

, Y)      :- parent_child(Z, X), parent_child(Z, Y).

ild(X, Y) :- father_child(X, Y).
ild(X, Y) :- mother_child(X, Y).
```

sibling(sally, erica)

- Falls under the category of Declarative Programming
- Comes with some great advantages. Suppor
  - Non-Determinism
  - Pattern-Matching
  - Meta-Programming
- Although it comes with neat features, it is not easy language to master.

Non-Determinism

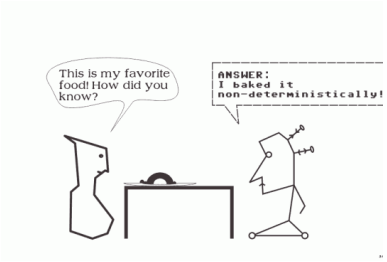
A non-deterministic language is one that can support, at various points in a program, different alternatives for program flow.

Pattern-Matching

As the name suggests, pattern-matching is the act of checking for a sequence.

Meta-Programming

Meta-programming is writing computer code that can manipulate, or write other programs, or even itself.

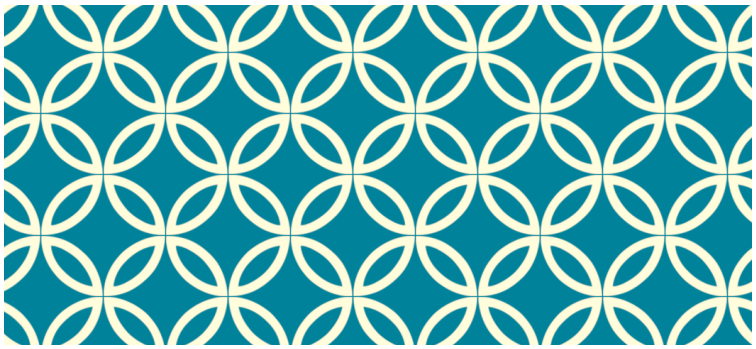


http://squidpony.com/chapter/when-fun-attacks/page/5/

```
other_child(trude, sally).

ather_child(tom, sally).
ather_child(tom, erica).
ather_child(mike, tom).

ibling(X, Y)      :- parent_child(Z, X), parent_child(Z, Y).

arent_child(X, Y) :- father_child(X, Y).
arent_child(X, Y) :- mother_child(X, Y).
```

query: ?- sibling(sally, erica)

1. To prove this query, Prolog first looks through the facts and rules, called clauses.
2. Matching clause is *sibling(X,Y)* :- *parent_child(Z,X), parent_child(Z,Y)*
3. Proving this query means proving the body.
4. Substitute sally for *X* to get *parent_child(Z,sally)*.
5. To prove above clause, Prolog looks through the clauses, and there is the *parent_child(X,Y)* rule, which is true if either *father_child(X,Y)* or *mother_child(X,Y)* is true. A choice point is created(non-determinism).
6. First check *father_child(X,Y)* by substituting *Z,sally* to *father_child(X,Y)*, creating *father_child(Z,sally)*.
7. Again look through the clauses. *father_child(Z,sally)* can be proven with the *father_child(tom,sally)* clause if Z is substituted with *tom*.
8. First part of the body is proven, and above steps are repeated with *parent_child(tom,erica)*.



FUNCTIONAL

Stephen

FUNCTIONAL PROGRAMING VS. IMPERATIVE PROGRAMING

Most programing is imperative

Functional used mainly in academia

Hardware optimized for imperative

Can be more productive using functional

FIBONACCI IN C (IMPERATIVE)

```
#include <iostream>

// Fibonacci numbers, imperative style
int fibonacci(int iterations)
{
        int first = 0, second = 1; //seed values
        for (int i = 0; i < iterations - 1; ++i)
        {
                int sum = first + second;
                first = second;
                second = sum;
        }
        return first;
}

int main()
{
        std::cout << fibonacci(10) << "\n";
        return 0;
}
```

FIBONACCI IN HASKELL (3 LINES)

```
-- Fibonacci numbers, functional style
-- describe an infinite list based on the recurrence relation
for Fibonacci numbers fibRecurrence first second = first : fibRecurrence second (first + second)
-- describe fibonacci list as fibRecurrence with initial values 0 and 1
fibonacci = fibRecurrence 0 1
-- describe action to print the 10th element of the fibonacci list
main = print (fibonacci !! 10)
```

FIBONACCI IN HASKELL (1 LINE)

```
fibonacci2 = 0:1:zipwith (+) fibonacci2 (tail fibonacci2)
```

KEY FUNCTIONAL PROGRAMMING CONCEPTS

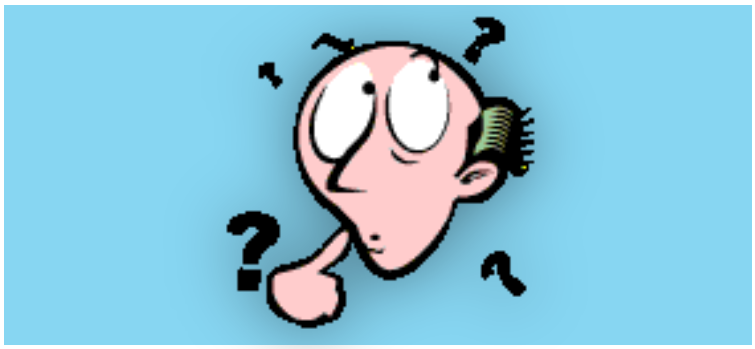
higher order functions

pure functions

recursion

strict vs. non strict

type inference



QUESTIONS?

We got answers!

...Well maybe

REFERENCES

http://ieeexplore.ieee.org.ezproxy.lib.ucalgary.ca/stamp/stamp.jsp?tp=&arnumber=728228

http://ieeexplore.ieee.org.ezproxy.lib.ucalgary.ca/stamp/stamp.jsp?tp=&arnumber=476556

http://cs.brown.edu/~adf/programming_languages.html

http://www.faircom.com/ace/enl_23_s08_t.php

http://ieeexplore.ieee.org.ezproxy.lib.ucalgary.ca/stamp/stamp.jsp?tp=&arnumber=4638280

Khan, Emdad H.; Al-A'ali; Girgis, Object Oriented Programming for Structured Procedural Programmers, University of Bahrain, 1995, p. 48-50.

Stannard, Kevin, Procedural vs Object Oriented Programming, http://objectorientedcoldfusion.org/procedural-vs-object-oriented.html, 2009-2011

Alltain, Alex, Lesson 4: Functions in C, http://www.cprogramming.com/tutorial/c/lesson4.html, 1997-2011.

Object Oriented Programming, http://www.cpp-home.com/archives/206.html, 2001-2013

Ahmad, Aftab; Talha, A Measurement Based Comparative Evaluation of Effectiveness of Object-Oriented Versus Conventional Procedural Programming Techniques and Languages, APSEC'02, p. 1-5.

Ishida, Toru; Sasaki; Fukuhara, Use of Procedural Programming Languages for Controlling Production Systems, NTT Communications and Information Processing Laboratories 1-2356, Take, Yokosuka-shi, 238-03, Japan, 1991, p. 71-75.

B. Smith, *AdvancED ActionScript 3.0: Design Patterns.*, Apress, 2011, p.428.

W. Schulz. 1998. Object-oriented programming. *SIGPLAN Fortran Forum* 17, 1 (April 1998), 11-19. DOI=10.1145/291709.291711 http://doi.acm.org/10.1145/291709.291711

I. Greenberg, *Processing: Creative Coding and Computational Art.*, Apress, 2007, pp. 301-336.

Java Tutorial, Retrieved February 21, 2013 from http://www.tutorialspoint.com/java/java_tutorial.pdf

Purcell, J. (2013). *Java Tutorial 5: Inheritance and Polymorphism.* Retrieved February 21, 2013 from http://www.coveofprogramming.com/javatutorial/java-tutorial-5-inheritance-and-polymorphism/

D'Andrea, R.J.; Gowda, R.G.; , "Object-oriented programming: concepts and languages," *Aerospace and Electronics Conference, 1990. NAECON 1990, Proceedings of the IEEE 1990 National* , vol., no., pp.634-640 vol.2, 21-25 May 1990 doi: 10.1109/NAECON.1990.112840

Hewitt, Carl. "Middle History of Logic Programming." n. page. Web. 20 Feb. 2013. <http://www.cs.unm.edu/~eschulte/classes/cs550/data/middle-hist-lp.pdf>.

Bruscoli, Paola. "Prolog Programming." University of Bath, United Kingdom, Bath. 2006. Lecture. <http://cs.bath.ac.uk/~pb275/CM20019/P/Prolog1.pdf>

Blackburn, Patrick, Johan Bos, and Kristina Striegnitz. "Learn Prolog Now!."http://www.learnprolognow.org. N.p.. Web. 27 Feb 2013. <http://www.learnprolognow.org/lpnpage.php?pageid=online>.

REFERENCES

"Prolog." *Wikipedia*. N.p.. Web. 27 Feb 2013. <http://en.wikipedia.org/wiki/Prolog>.

Hudak, Paul (September 1989). "Conception, evolution, and application of functional programming languages" (PDF). ACM *Computing Surveys* 21 (3): 359–411.

"Haskell in industry". *Haskell Wiki*. Retrieved 2009-08-26. "Haskell has a diverse range of use commercially, from aerospace and defense, to finance, to web startups, hardware design firms and lawnmower manufacturers."

"functools — Higher order functions and operations on callable objects". Python Software Foundation. 2011-07-31. Retrieved 2011-07-31.

Pawel Spiewak. "Implementing Persistent Vectors in Scala". Retrieved Apr 17, 2012.

"Which programs are fastest? | Computer Language Benchmarks Game".

Turner, D.A. (2004-07-28). "Total Functional Programming". *Journal of Universal Computer Science* 10 (7): 751–768. doi:10.3217/jucs-010-07-0751.

"Who uses Erlang for product development?". *Frequently asked questions about Erlang*. Retrieved 2007-08-05.

Laurikari, Ville (2007). "Functional Programming in Communications Security.". CUFP 2007. Retrieved 2009-08-29.

Haskell Brooks Curry; Robert Feys (1958). Combinatory Logic. North-Holland Publishing Company. Retrieved 10 February 2013.

www.cpe.ac.th/~parul/30183/lec/OOP.ppt