

# Prototype-based 基于原型编程

- 是面向对象编程的一种方式。没有class化的，直接使用对象。又叫，基于实例的编程。
- 主流的语言是 Javascript
- 比较
  - 基于类的面向对象，关于于类和子类的模型。
  - 基于原型的面向对象，关注一系列对象实例的行为，而之后才关心如何将对象划分到最近的使用方式相似的原型对象，而不是分成类。
- 很多基于原型的系统提倡运行时原型的修改，而基于类的面向对象系统只有动态语言允许类在运行时被修改（Common Lisp, Dylan, Objective-C, Perl, Python, Ruby, or Smalltalk）。

# 一个简单的示例

```
//Define human class
var Person = function (fullName) {
  this.fullName = fullName;
  // Add a couple of methods to Person.prototype
  this.speak = function(){
    console.log("I speak English!");
  };
  this.introduction = function(){
    console.log("Hi, I am " + this.fullName);
  };
}

//Define Student class
var Student = function(fullName, school, courses) {

  Person.call(this, fullName);

  // Initialize our Student properties
  this.school = school;
  this.courses = courses;
  // override the "introduction" method
  this.introduction = function(){
    console.log("Hi, I am " + this.fullName +
      ". I am a student of " + this.school +
      ", I study " + this.courses + ".");
  };
  // Add a "exams" method
  this.takeExams = function(){
    console.log("This is my exams time!");
  };
};
```

```
// Create a Student.prototype object that inherits
// from Person.prototype.
Student.prototype = Object.create(Person.prototype);

// Set the "constructor" property to refer to Student
Student.prototype.constructor = Student;

var student = new Student("Hao Chen",
  "XYZ University",
  "Computer Science");

student.introduction();
student.speak();
student.takeExams();

// Check that instanceof works correctly
console.log(student instanceof Person); // true
console.log(student instanceof Student); // true
```

- 担心：正确性、安全性、可预测性以及效率
- 在 ECMAScript 标准的第四版开始寻求使 JavaScript 提供基于类的构造，且 ECMAScript 第六版有提供"class"(类)作为原有的原型架构之上的语法糖，提供建构对象与处理继承时的另一种语法



# GO 语言的委托模式

---

# Go语言的接口和委托

```
type Widget struct {
    X, Y int
}

type Label struct {
    Widget      // Embedding (delegation)
    Text string // Aggregation
    X int       // Override
}

func (label Label) Paint() {
    // [0xc4200141e0] - Label.Paint("State")
    fmt.Printf("[%p] - Label.Paint(%q)\n",
        &label, label.Text)
}
```

```
label := Label{Widget{10, 10}, "State", 100}

// X=100, Y=10, Text=State, Widget.X=10
fmt.Printf("X=%d, Y=%d, Text=%s Widget.X=%d\n",
    label.X, label.Y, label.Text,
    label.Widget.X)
fmt.Println()
// {Widget:{X:10 Y:10} Text:State X:100}
// {{10 10} State 100}
fmt.Printf("%+v\n%v\n", label, label)
```

- 通过直接写入类型达到委托的效果
- 如果有成员变量重名，则需要解决冲突

# Go语言的接口和委托

```
type Button struct {
    Label // Embedding (delegation)
}

func NewButton(x, y int, text string) Button {
    return Button{Label{Widget{x, y}, text, x}}
}

func (button Button) Paint() { // Override
    fmt.Printf("[%p] - Button.Paint(%q)\n",
        &button, button.Text)
}

func (button Button) Click() {
    fmt.Printf("[%p] - Button.Click()\n", &button)
}
```

```
type ListBox struct {
    Widget // Embedding (delegation)
    Texts []string // Aggregation
    Index int // Aggregation
}

func (listBox ListBox) Paint() {
    fmt.Printf("[%p] - ListBox.Paint(%q)\n",
        &listBox, listBox.Texts)
}

func (listBox ListBox) Click() {
    fmt.Printf("[%p] - ListBox.Click()\n", &listBox)
}
```

```
type Painter interface {
    Paint()
}

type Clicker interface {
    Click()
}
```

```
for _, painter := range []Painter{label, listBox, button1, button2} {
    painter.Paint()
}

for _, widget := range []interface{}{label, listBox, button1, button2} {
    if clicker, ok := widget.(Clicker); ok {
        clicker.Click()
    }
}
```

```

type IntSet struct {
    data map[int]bool
}

func NewIntSet() IntSet {
    return IntSet{make(map[int]bool)}
}

func (set *IntSet) Add(x int) {
    set.data[x] = true
}

func (set *IntSet) Delete(x int) {
    delete(set.data, x)
}

func (set *IntSet) Contains(x int) bool {
    return set.data[x]
}

```

```

type UndoableIntSet struct { // Poor style
    IntSet // Embedding (delegation)
    functions []func()
}

func NewUndoableIntSet() UndoableIntSet {
    return UndoableIntSet{NewIntSet(), nil}
}

func (set *UndoableIntSet) Add(x int) { // Override
    if !set.Contains(x) {
        set.data[x] = true
        set.functions = append(set.functions, func() { set.Delete(x) })
    } else {
        set.functions = append(set.functions, nil)
    }
}

func (set *UndoableIntSet) Delete(x int) { // Override
    if set.Contains(x) {
        delete(set.data, x)
        set.functions = append(set.functions, func() { set.Add(x) })
    } else {
        set.functions = append(set.functions, nil)
    }
}

```

- 通过委托扩展原有的功能
- UndoableIntSet 增加了一个Undo的方法

```

func (set *UndoableIntSet) Undo() error {
    if len(set.functions) == 0 {
        return errors.New("No functions to undo")
    }
    index := len(set.functions) - 1
    if function := set.functions[index]; function != nil {
        function()
        set.functions[index] = nil // Free closure for garbage collection
    }
    set.functions = set.functions[:index]
    return nil
}

```

```

type Undo []func()

func (undo *Undo) Add(function func()) {
    *undo = append(*undo, function)
}

func (undo *Undo) Undo() error {
    functions := *undo
    if len(functions) == 0 {
        return errors.New("No functions to undo")
    }
    index := len(functions) - 1
    if function := functions[index]; function != nil {
        function()
        functions[index] = nil // Free closure for garbage collection
    }
    *undo = functions[:index]
    return nil
}

```

- 单独实现一个Undo 的泛型
- 这样可以直接委托给其它结构，从而完成这个功能
- 不足，每个结构都需要实现类似的Undo协议

```

type IntSet struct {
    data map[int]bool
    undo Undo
}

func NewIntSet() IntSet {
    return IntSet{data: make(map[int]bool)}
}

func (set *IntSet) Add(x int) {
    if !set.Contains(x) {
        set.data[x] = true
        set.undo.Add(func() { set.Delete(x) })
    } else {
        set.undo.Add(nil)
    }
}

func (set *IntSet) Delete(x int) {
    if set.Contains(x) {
        delete(set.data, x)
        set.undo.Add(func() { set.Add(x) })
    } else {
        set.undo.Add(nil)
    }
}

func (set *IntSet) Undo() error {
    return set.undo.Undo()
}

func (set *IntSet) Contains(x int) bool {
    return set.data[x]
}

```



# 编程的本质

---

# 程序的本质

- Pascal语言之父Niklaus Wirth在70年代提出：
  - Program = Data Structure + Algorithm
- 随后逻辑学家和计算机科学家R Kowalski进一步提出：
  - Algorithm = Logic + Control
- **程序的复杂性有两个：**
  - 一个是代码需要处理的逻辑，也就是我们说的业务逻辑
  - 另一个是控制，控制或组织代码完成复杂的逻辑

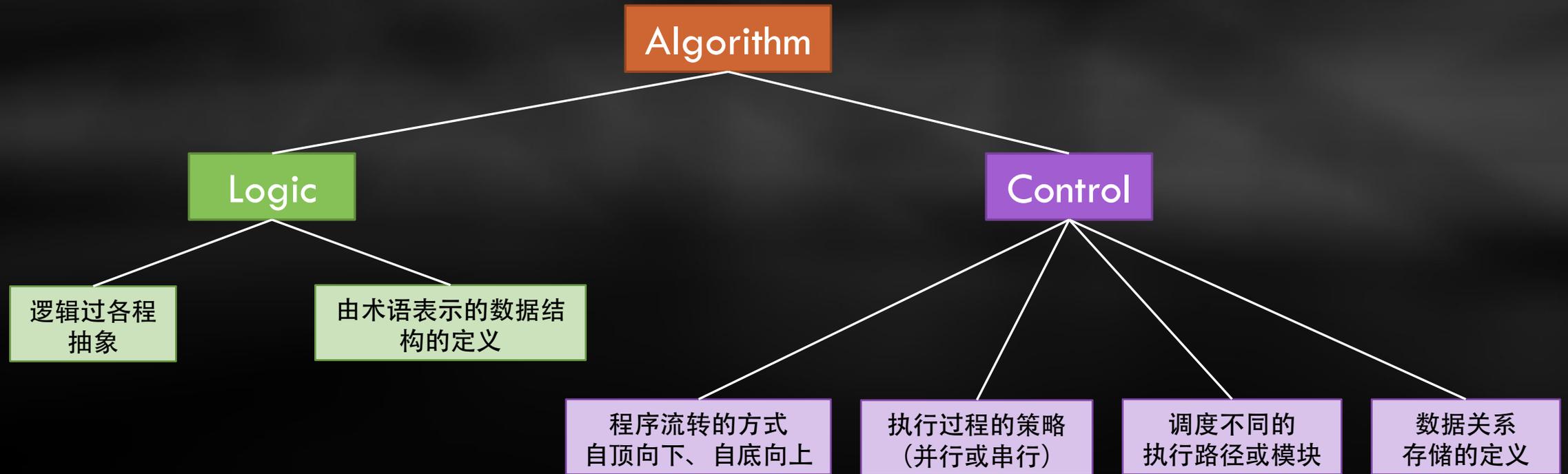
# 示例：Control Logic 混在一起

- 通配符匹配

- isMatch("aa","a") → false
- isMatch("aa","aa") → true
- isMatch("aaa","aa") → false
- isMatch("aa","\*") → true
- isMatch("aa","a\*") → true
- isMatch("ab","?\*") → true
- isMatch("aab","c\*a\*b") → false

```
bool isMatch(const char *s, const char *p) {
    const char *last_s = NULL;
    const char *last_p = NULL;
    while( *s != '\0' ){
        if (*p=='*'){
            p++;
            if (*p=='\0') return true;
            last_s = s;
            last_p = p;
        }else if (*p=='?' || *s == *p){
            s++; p++;
        }else if (last_s != NULL){
            p = last_p;
            s = ++last_s;
        }else{
            return false;
        }
    }
    while (*p == '*') p++;
    return *p == '\0';
}
```

# 程序 的本质

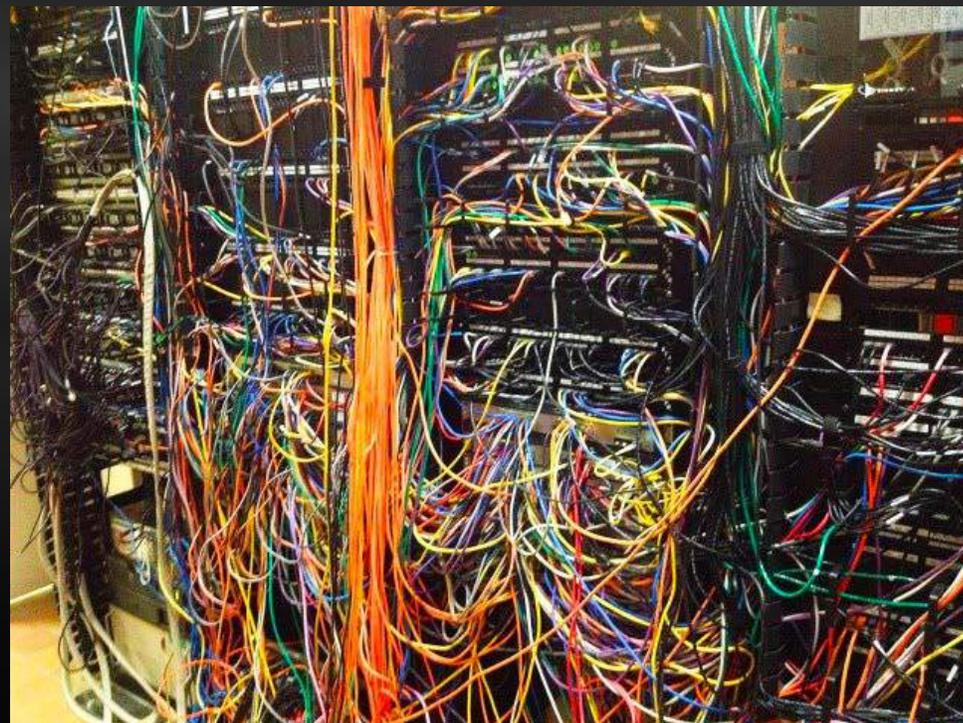


- **Logic 部分才是真正有意义的 - What**
- **Control部分只是影响Logic部分的效率 - How**

# 程序的复杂度

- 业务逻辑的复杂度决定了代码的复杂度
- 控制逻辑的复杂度 + 业务逻辑的复杂度 ==> 程序代码的混乱不堪
- 绝大多数程序复杂混乱的根本原因：

**业务逻辑 与 控制逻辑 的 耦合**



# 如何分离 Control 和 Logic

- **State Machine**
  - 状态定义,
  - 状态变迁条件
  - 状态的action
- **DSL – Domain Specific Language**
  - HTML, SQL, Unix Shell Script, AWK, 正则表达式……
- **编程范式**
  - 面向对象 - 委托、策略、桥接、修饰、IoC/DIP、MVC……
  - 函数式编程 - 修饰、管道、拼装
  - 逻辑推导式编程 - Prolog

# 一个简单的示例

```
function check_form() {
  var name = $('#name').val();
  if (null == name || name.length <= 3) {
    return { status : 1, message: 'Invalid name' };
  }
  var password1 = $('#password1').val();
  if (null == password1 || password1.length <= 8) {
    return { status : 2, message: 'Invalid password' };
  }
  var password2 = $('#password2').val();
  if (password2 != password1) {
    return { status : 3, message: 'Password mismatch' };
  }
  var email = $('#email').val();
  if (check_email_format(email)) {
    return { status : 4, message: 'Invalid email' };
  }
  return { status : 0, message: 'OK' };
}
```

```
var meta_create_user = {
  form_id : 'create_user',
  fields : [
    { id : 'name', type : 'text', min_length : 3 },
    { id : 'password1', type : 'password', min_length : 8 },
    { id : 'password2', type : 'password', min_length : 8 },
    { id : 'email', type : 'email' }
  ]
};

var r = check_form(meta_create_user);
```



# 逻辑编程范式

---

# 逻辑编程范式

- 逻辑编程的要点是将正规的逻辑风格带入电脑程序设计之中。
- 逻辑编程建立了描述一个问题里的世界的逻辑模型。
- 逻辑编程的目标是对它的模型建立新的陈述。
  - 通过陈述事实——因果关系
  - 程序自动推导出相关的逻辑

```
program mortal(X) :- person(X).  
  
person(Socrates).  
person(Plato).  
person(Aristotle).  
  
mortal_report:-  
write('Known mortals are:'),nl, mortal(X),  
write(X),nl,  
fail.
```

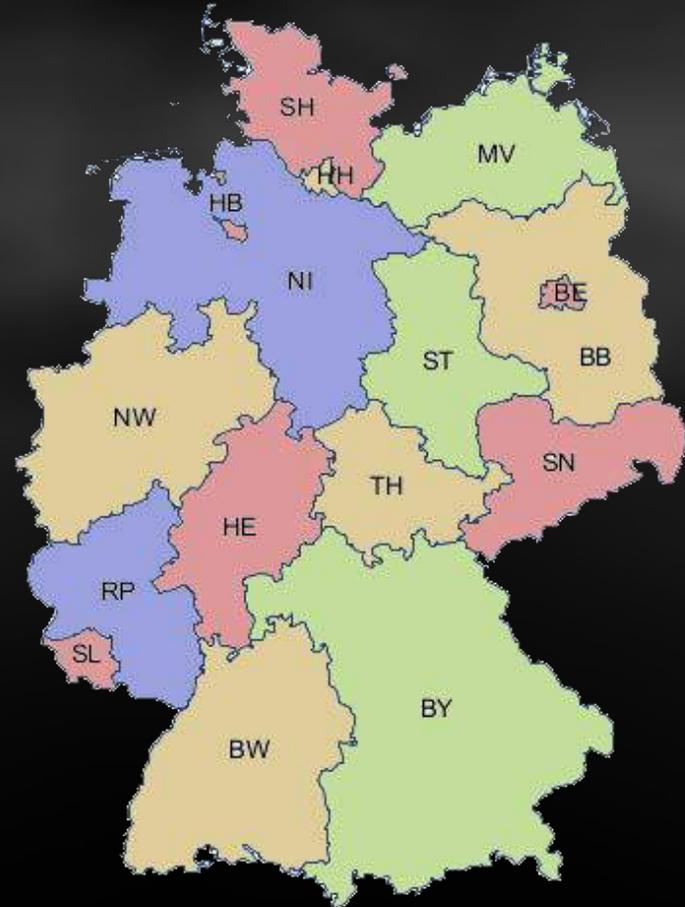
# 地图四色问题 - Prolog

```
/* 定义四种颜色 */
color(red).
color(green).
color(blue).
color(yellow).

/* 定义逻辑 */
neighbor(StateAColor, StateBColor) :-
    color(StateAColor),
    color(StateBColor),
    StateAColor \= StateBColor.
/* \= is the not equal operator */

/* 定义地图 BW 和 BY 相邻 */
germany(BW, BY) :- neighbor(BW, BY).

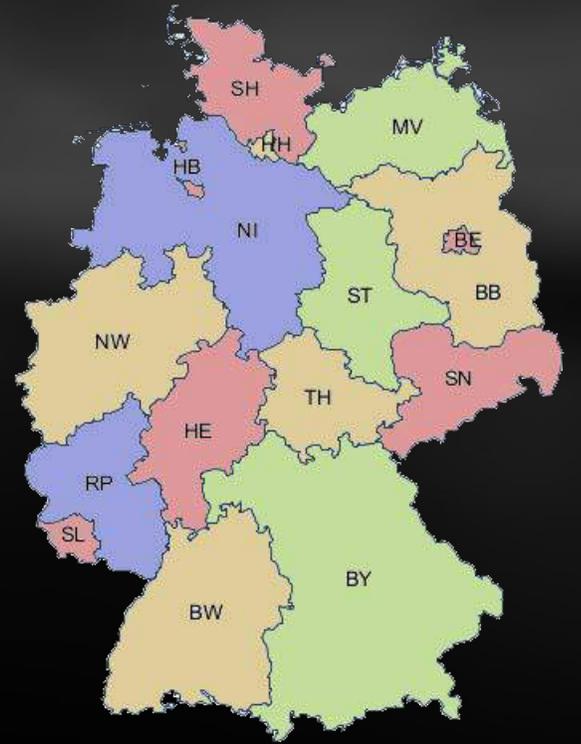
/* 定义地图 BW 和BY RP、HE相邻关系 */
germany(BW, BY, RP, HE) :-
    neighbor(BW, BY),
    neighbor(BW, RP),
    neighbor(BW, HE).
```



# 地图四色问题 - Prolog

```
/* 定义整个地图的相邻关系 */
germany(SH, MV, HH, HB, NI, ST, BE, BB, SN, NW, HE, TH, RP, SL, BW, BY) :-
  neighbor(SH, NI), neighbor(SH, HH), neighbor(SH, MV),
  neighbor(HH, NI),
  neighbor(MV, NI), neighbor(MV, BB),
  neighbor(NI, HB), neighbor(NI, BB), neighbor(NI, ST), neighbor(NI, TH),
  neighbor(NI, HE), neighbor(NI, NW),
  neighbor(ST, BB), neighbor(ST, SN), neighbor(ST, TH),
  neighbor(BB, BE), neighbor(BB, SN),
  neighbor(NW, HE), neighbor(NW, RP),
  neighbor(SN, TH), neighbor(SN, BY),
  neighbor(RP, SL), neighbor(RP, HE), neighbor(RP, BW),
  neighbor(HE, BW), neighbor(HE, TH), neighbor(HE, BY),
  neighbor(TH, BY),
  neighbor(BW, BY).

/* 推导求值 */
?- germany(SH, MV, HH, HB, NI, ST, BE, BB, SN, NW, HE, TH, RP, SL, BW, BY).
```





# 程序世界里的编程范式

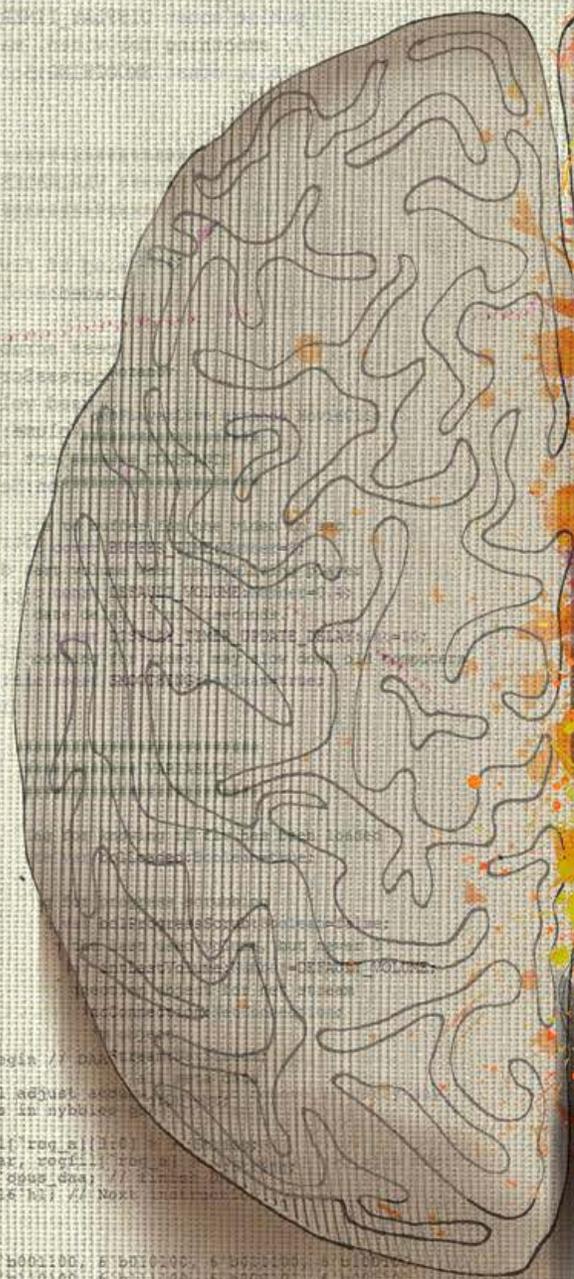
---

```
utils:File;
resStream:InputStream;
void fillMetaData
objInfo:Object;
tv file
strSource:String;
videoFileName:String;
obj_pic:MovieClip;
```

# Left brain

I am the left brain.  
I am a scientist. A mathematician.  
I love the familiar. I categorize. I am accurate. Linear.  
Analytical. Strategic. I am practical.  
Always in control. A master of words and language.  
Realistic. I calculate equations and play with numbers.  
I am order. I am logic.  
I know exactly who I am.

理性分析型  
喜欢数据证据  
线性思维  
陷入细节  
具体化的



# Right brain

I am the right brain.  
I am creativity. A free spirit. I am passion.  
Yearning. Sensuality. I am the sound of roaring laughter.  
I am taste. The feeling of sand beneath bare feet.  
I am movement. Vivid colors.  
I am the urge to paint on an empty canvas.  
I am boundless imagination. Art. Poetry. I sense. I feel.  
I am everything I wanted to be.

直觉型  
想像力  
非线性  
宏观思维  
抽象化的



```
5 210011: begin // ON
// decimal request
// results in nybble
if (code[1] == reg[1])
{
  auopra = reg[1];
  pc << pc+8; // Next
}
end
6 200010: 6 200110: 6 201010: 6 202110: 6 203110:
6 204110: 6 205110: 6 206110: 6 207110: 6 208110:
6 209110: 6 210110: 6 211110: 6 212110: 6 213110:
6 214110: 6 215110: 6 216110: 6 217110: 6 218110:
6 219110: 6 220110: 6 221110: 6 222110: 6 223110:
6 224110: 6 225110: 6 226110: 6 227110: 6 228110:
6 229110: 6 230110: 6 231110: 6 232110: 6 233110:
6 234110: 6 235110: 6 236110: 6 237110: 6 238110:
6 239110: 6 240110: 6 241110: 6 242110: 6 243110:
6 244110: 6 245110: 6 246110: 6 247110: 6 248110:
6 249110: 6 250110: 6 251110: 6 252110: 6 253110:
6 254110: 6 255110: 6 256110: 6 257110: 6 258110:
6 259110: 6 260110: 6 261110: 6 262110: 6 263110:
6 264110: 6 265110: 6 266110: 6 267110: 6 268110:
6 269110: 6 270110: 6 271110: 6 272110: 6 273110:
6 274110: 6 275110: 6 276110: 6 277110: 6 278110:
6 279110: 6 280110: 6 281110: 6 282110: 6 283110:
6 284110: 6 285110: 6 286110: 6 287110: 6 288110:
6 289110: 6 290110: 6 291110: 6 292110: 6 293110:
6 294110: 6 295110: 6 296110: 6 297110: 6 298110:
6 299110: 6 300110: 6 301110: 6 302110: 6 303110:
6 304110: 6 305110: 6 306110: 6 307110: 6 308110:
6 309110: 6 310110: 6 311110: 6 312110: 6 313110:
6 314110: 6 315110: 6 316110: 6 317110: 6 318110:
6 319110: 6 320110: 6 321110: 6 322110: 6 323110:
6 324110: 6 325110: 6 326110: 6 327110: 6 328110:
6 329110: 6 330110: 6 331110: 6 332110: 6 333110:
6 334110: 6 335110: 6 336110: 6 337110: 6 338110:
6 339110: 6 340110: 6 341110: 6 342110: 6 343110:
6 344110: 6 345110: 6 346110: 6 347110: 6 348110:
6 349110: 6 350110: 6 351110: 6 352110: 6 353110:
6 354110: 6 355110: 6 356110: 6 357110: 6 358110:
6 359110: 6 360110: 6 361110: 6 362110: 6 363110:
6 364110: 6 365110: 6 366110: 6 367110: 6 368110:
6 369110: 6 370110: 6 371110: 6 372110: 6 373110:
6 374110: 6 375110: 6 376110: 6 377110: 6 378110:
6 379110: 6 380110: 6 381110: 6 382110: 6 383110:
6 384110: 6 385110: 6 386110: 6 387110: 6 388110:
6 389110: 6 390110: 6 391110: 6 392110: 6 393110:
6 394110: 6 395110: 6 396110: 6 397110: 6 398110:
6 399110: 6 400110: 6 401110: 6 402110: 6 403110:
6 404110: 6 405110: 6 406110: 6 407110: 6 408110:
6 409110: 6 410110: 6 411110: 6 412110: 6 413110:
6 414110: 6 415110: 6 416110: 6 417110: 6 418110:
6 419110: 6 420110: 6 421110: 6 422110: 6 423110:
6 424110: 6 425110: 6 426110: 6 427110: 6 428110:
6 429110: 6 430110: 6 431110: 6 432110: 6 433110:
6 434110: 6 435110: 6 436110: 6 437110: 6 438110:
6 439110: 6 440110: 6 441110: 6 442110: 6 443110:
6 444110: 6 445110: 6 446110: 6 447110: 6 448110:
6 449110: 6 450110: 6 451110: 6 452110: 6 453110:
6 454110: 6 455110: 6 456110: 6 457110: 6 458110:
6 459110: 6 460110: 6 461110: 6 462110: 6 463110:
6 464110: 6 465110: 6 466110: 6 467110: 6 468110:
6 469110: 6 470110: 6 471110: 6 472110: 6 473110:
6 474110: 6 475110: 6 476110: 6 477110: 6 478110:
6 479110: 6 480110: 6 481110: 6 482110: 6 483110:
6 484110: 6 485110: 6 486110: 6 487110: 6 488110:
6 489110: 6 490110: 6 491110: 6 492110: 6 493110:
6 494110: 6 495110: 6 496110: 6 497110: 6 498110:
6 499110: 6 500110: 6 501110: 6 502110: 6 503110:
6 504110: 6 505110: 6 506110: 6 507110: 6 508110:
6 509110: 6 510110: 6 511110: 6 512110: 6 513110:
6 514110: 6 515110: 6 516110: 6 517110: 6 518110:
6 519110: 6 520110: 6 521110: 6 522110: 6 523110:
6 524110: 6 525110: 6 526110: 6 527110: 6 528110:
6 529110: 6 530110: 6 531110: 6 532110: 6 533110:
6 534110: 6 535110: 6 536110: 6 537110: 6 538110:
6 539110: 6 540110: 6 541110: 6 542110: 6 543110:
6 544110: 6 545110: 6 546110: 6 547110: 6 548110:
6 549110: 6 550110: 6 551110: 6 552110: 6 553110:
6 554110: 6 555110: 6 556110: 6 557110: 6 558110:
6 559110: 6 560110: 6 561110: 6 562110: 6 563110:
6 564110: 6 565110: 6 566110: 6 567110: 6 568110:
6 569110: 6 570110: 6 571110: 6 572110: 6 573110:
6 574110: 6 575110: 6 576110: 6 577110: 6 578110:
6 579110: 6 580110: 6 581110: 6 582110: 6 583110:
6 584110: 6 585110: 6 586110: 6 587110: 6 588110:
6 589110: 6 590110: 6 591110: 6 592110: 6 593110:
6 594110: 6 595110: 6 596110: 6 597110: 6 598110:
6 599110: 6 600110: 6 601110: 6 602110: 6 603110:
6 604110: 6 605110: 6 606110: 6 607110: 6 608110:
6 609110: 6 610110: 6 611110: 6 612110: 6 613110:
6 614110: 6 615110: 6 616110: 6 617110: 6 618110:
6 619110: 6 620110: 6 621110: 6 622110: 6 623110:
6 624110: 6 625110: 6 626110: 6 627110: 6 628110:
6 629110: 6 630110: 6 631110: 6 632110: 6 633110:
6 634110: 6 635110: 6 636110: 6 637110: 6 638110:
6 639110: 6 640110: 6 641110: 6 642110: 6 643110:
6 644110: 6 645110: 6 646110: 6 647110: 6 648110:
6 649110: 6 650110: 6 651110: 6 652110: 6 653110:
6 654110: 6 655110: 6 656110: 6 657110: 6 658110:
6 659110: 6 660110: 6 661110: 6 662110: 6 663110:
6 664110: 6 665110: 6 666110: 6 667110: 6 668110:
6 669110: 6 670110: 6 671110: 6 672110: 6 673110:
6 674110: 6 675110: 6 676110: 6 677110: 6 678110:
6 679110: 6 680110: 6 681110: 6 682110: 6 683110:
6 684110: 6 685110: 6 686110: 6 687110: 6 688110:
6 689110: 6 690110: 6 691110: 6 692110: 6 693110:
6 694110: 6 695110: 6 696110: 6 697110: 6 698110:
6 699110: 6 700110: 6 701110: 6 702110: 6 703110:
6 704110: 6 705110: 6 706110: 6 707110: 6 708110:
6 709110: 6 710110: 6 711110: 6 712110: 6 713110:
6 714110: 6 715110: 6 716110: 6 717110: 6 718110:
6 719110: 6 720110: 6 721110: 6 722110: 6 723110:
6 724110: 6 725110: 6 726110: 6 727110: 6 728110:
6 729110: 6 730110: 6 731110: 6 732110: 6 733110:
6 734110: 6 735110: 6 736110: 6 737110: 6 738110:
6 739110: 6 740110: 6 741110: 6 742110: 6 743110:
6 744110: 6 745110: 6 746110: 6 747110: 6 748110:
6 749110: 6 750110: 6 751110: 6 752110: 6 753110:
6 754110: 6 755110: 6 756110: 6 757110: 6 758110:
6 759110: 6 760110: 6 761110: 6 762110: 6 763110:
6 764110: 6 765110: 6 766110: 6 767110: 6 768110:
6 769110: 6 770110: 6 771110: 6 772110: 6 773110:
6 774110: 6 775110: 6 776110: 6 777110: 6 778110:
6 779110: 6 780110: 6 781110: 6 782110: 6 783110:
6 784110: 6 785110: 6 786110: 6 787110: 6 788110:
6 789110: 6 790110: 6 791110: 6 792110: 6 793110:
6 794110: 6 795110: 6 796110: 6 797110: 6 798110:
6 799110: 6 800110: 6 801110: 6 802110: 6 803110:
6 804110: 6 805110: 6 806110: 6 807110: 6 808110:
6 809110: 6 810110: 6 811110: 6 812110: 6 813110:
6 814110: 6 815110: 6 816110: 6 817110: 6 818110:
6 819110: 6 820110: 6 821110: 6 822110: 6 823110:
6 824110: 6 825110: 6 826110: 6 827110: 6 828110:
6 829110: 6 830110: 6 831110: 6 832110: 6 833110:
6 834110: 6 835110: 6 836110: 6 837110: 6 838110:
6 839110: 6 840110: 6 841110: 6 842110: 6 843110:
6 844110: 6 845110: 6 846110: 6 847110: 6 848110:
6 849110: 6 850110: 6 851110: 6 852110: 6 853110:
6 854110: 6 855110: 6 856110: 6 857110: 6 858110:
6 859110: 6 860110: 6 861110: 6 862110: 6 863110:
6 864110: 6 865110: 6 866110: 6 867110: 6 868110:
6 869110: 6 870110: 6 871110: 6 872110: 6 873110:
6 874110: 6 875110: 6 876110: 6 877110: 6 878110:
6 879110: 6 880110: 6 881110: 6 882110: 6 883110:
6 884110: 6 885110: 6 886110: 6 887110: 6 888110:
6 889110: 6 890110: 6 891110: 6 892110: 6 893110:
6 894110: 6 895110: 6 896110: 6 897110: 6 898110:
6 899110: 6 900110: 6 901110: 6 902110: 6 903110:
6 904110: 6 905110: 6 906110: 6 907110: 6 908110:
6 909110: 6 910110: 6 911110: 6 912110: 6 913110:
6 914110: 6 915110: 6 916110: 6 917110: 6 918110:
6 919110: 6 920110: 6 921110: 6 922110: 6 923110:
6 924110: 6 925110: 6 926110: 6 927110: 6 928110:
6 929110: 6 930110: 6 931110: 6 932110: 6 933110:
6 934110: 6 935110: 6 936110: 6 937110: 6 938110:
6 939110: 6 940110: 6 941110: 6 942110: 6 943110:
6 944110: 6 945110: 6 946110: 6 947110: 6 948110:
6 949110: 6 950110: 6 951110: 6 952110: 6 953110:
6 954110: 6 955110: 6 956110: 6 957110: 6 958110:
6 959110: 6 960110: 6 961110: 6 962110: 6 963110:
6 964110: 6 965110: 6 966110: 6 967110: 6 968110:
6 969110: 6 970110: 6 971110: 6 972110: 6 973110:
6 974110: 6 975110: 6 976110: 6 977110: 6 978110:
6 979110: 6 980110: 6 981110: 6 982110: 6 983110:
6 984110: 6 985110: 6 986110: 6 987110: 6 988110:
6 989110: 6 990110: 6 991110: 6 992110: 6 993110:
6 994110: 6 995110: 6 996110: 6 997110: 6 998110:
6 999110: 6 1000110: 6 1001110: 6 1002110: 6 1003110:
6 1004110: 6 1005110: 6 1006110: 6 1007110: 6 1008110:
6 1009110: 6 1010110: 6 1011110: 6 1012110: 6 1013110:
6 1014110: 6 1015110: 6 1016110: 6 1017110: 6 1018110:
6 1019110: 6 1020110: 6 1021110: 6 1022110: 6 1023110:
6 1024110: 6 1025110: 6 1026110: 6 1027110: 6 1028110:
6 1029110: 6 1030110: 6 1031110: 6 1032110: 6 1033110:
6 1034110: 6 1035110: 6 1036110: 6 1037110: 6 1038110:
6 1039110: 6 1040110: 6 1041110: 6 1042110: 6 1043110:
6 1044110: 6 1045110: 6 1046110: 6 1047110: 6 1048110:
6 1049110: 6 1050110: 6 1051110: 6 1052110: 6 1053110:
6 1054110: 6 1055110: 6 1056110: 6 1057110: 6 1058110:
6 1059110: 6 1060110: 6 1061110: 6 1062110: 6 1063110:
6 1064110: 6 1065110: 6 1066110: 6 1067110: 6 1068110:
6 1069110: 6 1070110: 6 1071110: 6 1072110: 6 1073110:
6 1074110: 6 1075110: 6 1076110: 6 1077110: 6 1078110:
6 1079110: 6 1080110: 6 1081110: 6 1082110: 6 1083110:
6 1084110: 6 1085110: 6 1086110: 6 1087110: 6 1088110:
6 1089110: 6 1090110: 6 1091110: 6 1092110: 6 1093110:
6 1094110: 6 1095110: 6 1096110: 6 1097110: 6 1098110:
6 1099110: 6 1100110: 6 1101110: 6 1102110: 6 1103110:
6 1104110: 6 1105110: 6 1106110: 6 1107110: 6 1108110:
6 1109110: 6 1110110: 6 1111110: 6 1112110: 6 1113110:
6 1114110: 6 1115110: 6 1116110: 6 1117110: 6 1118110:
6 1119110: 6 1120110: 6 1121110: 6 1122110: 6 1123110:
6 1124110: 6 1125110: 6 1126110: 6 1127110: 6 1128110:
6 1129110: 6 1130110: 6 1131110: 6 1132110: 6 1133110:
6 1134110: 6 1135110: 6 1136110: 6 1137110: 6 1138110:
6 1139110: 6 1140110: 6 1141110: 6 1142110: 6 1143110:
6 1144110: 6 1145110: 6 1146110: 6 1147110: 6 1148110:
6 1149110: 6 1150110: 6 1151110: 6 1152110: 6 1153110:
6 1154110: 6 1155110: 6 1156110: 6 1157110: 6 1158110:
6 1159110: 6 1160110: 6 1161110: 6 1162110: 6 1163110:
6 1164110: 6 1165110: 6 1166110: 6 1167110: 6 1168110:
6 1169110: 6 1170110: 6 1171110: 6 1172110: 6 1173110:
6 1174110: 6 1175110: 6 1176110: 6 1177110: 6 1178110:
6 1179110: 6 1180110: 6 1181110: 6 1182110: 6 1183110:
6 1184110: 6 1185110: 6 1186110: 6 1187110: 6 1188110:
6 1189110: 6 1190110: 6 1191110: 6 1192110: 6 1193110:
6 1194110: 6 1195110: 6 1196110: 6 1197110: 6 1198110:
6 1199110: 6 1200110: 6 1201110: 6 1202110: 6 1203110:
6 1204110: 6 1205110: 6 1206110: 6 1207110: 6 1208110:
6 1209110: 6 1210110: 6 1211110: 6 1212110: 6 1213110:
6 1214110: 6 1215110: 6 1216110: 6 1217110: 6 1218110:
6 1219110: 6 1220110: 6 1221110: 6 1222110: 6 1223110:
6 1224110: 6 1225110: 6 1226110: 6 1227110: 6 1228110:
6 1229110: 6 1230110: 6 1231110: 6 1232110: 6 1233110:
6 1234110: 6 1235110: 6 1236110: 6 1237110: 6 1238110:
6 1239110: 6 1240110: 6 1241110: 6 1242110: 6 1243110:
6 1244110: 6 1245110: 6 1246110: 6 1247110: 6 1248110:
6 1249110: 6 1250110: 6 1251110: 6 1252110: 6 1253110:
6 1254110: 6 1255110: 6 1256110: 6 1257110: 6 1258110:
6 1259110: 6 1260110: 6 1261110: 6 1262110: 6 1263110:
6 1264110: 6 1265110: 6 1266110: 6 1267110: 6 1268110:
6 1269110: 6 1270110: 6 1271110: 6 1272110: 6 1273110:
6 1274110: 6 1275110: 6 1276110: 6 1277110: 6 1278110:
6 1279110: 6 1280110: 6 1281110: 6 1282110: 6 1283110:
6 1284110: 6 1285110: 6 1286110: 6 1287110: 6 1288110:
6 1289110: 6 1290110: 6 1291110: 6 1292110: 6 1293110:
6 1294110: 6 1295110: 6 1296110: 6 1297110: 6 1298110:
6 1299110: 6 1300110: 6 1301110: 6 1302110: 6 1303110:
6 1304110: 6 1305110: 6 1306110: 6 1307110: 6 1308110:
6 1309110: 6 1310110: 6 1311110: 6 1312110: 6 1313110:
6 1314110: 6 1315110: 6 1316110: 6 1317110: 6 1318110:
6 1319110: 6 1320110: 6 1321110: 6 1322110: 6 1323110:
6 1324110: 6 1325110: 6 13261
```

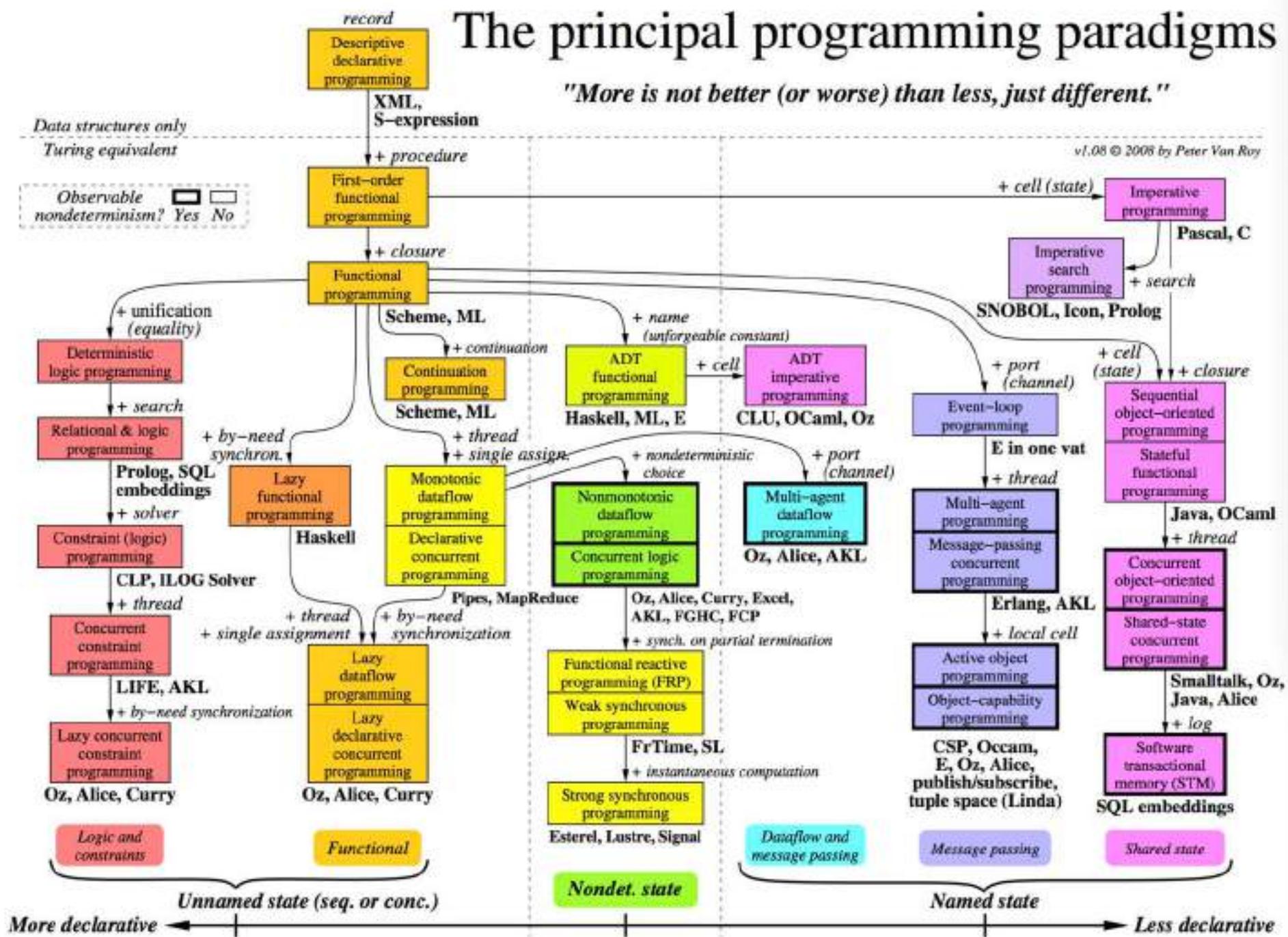
# The principal programming paradigms

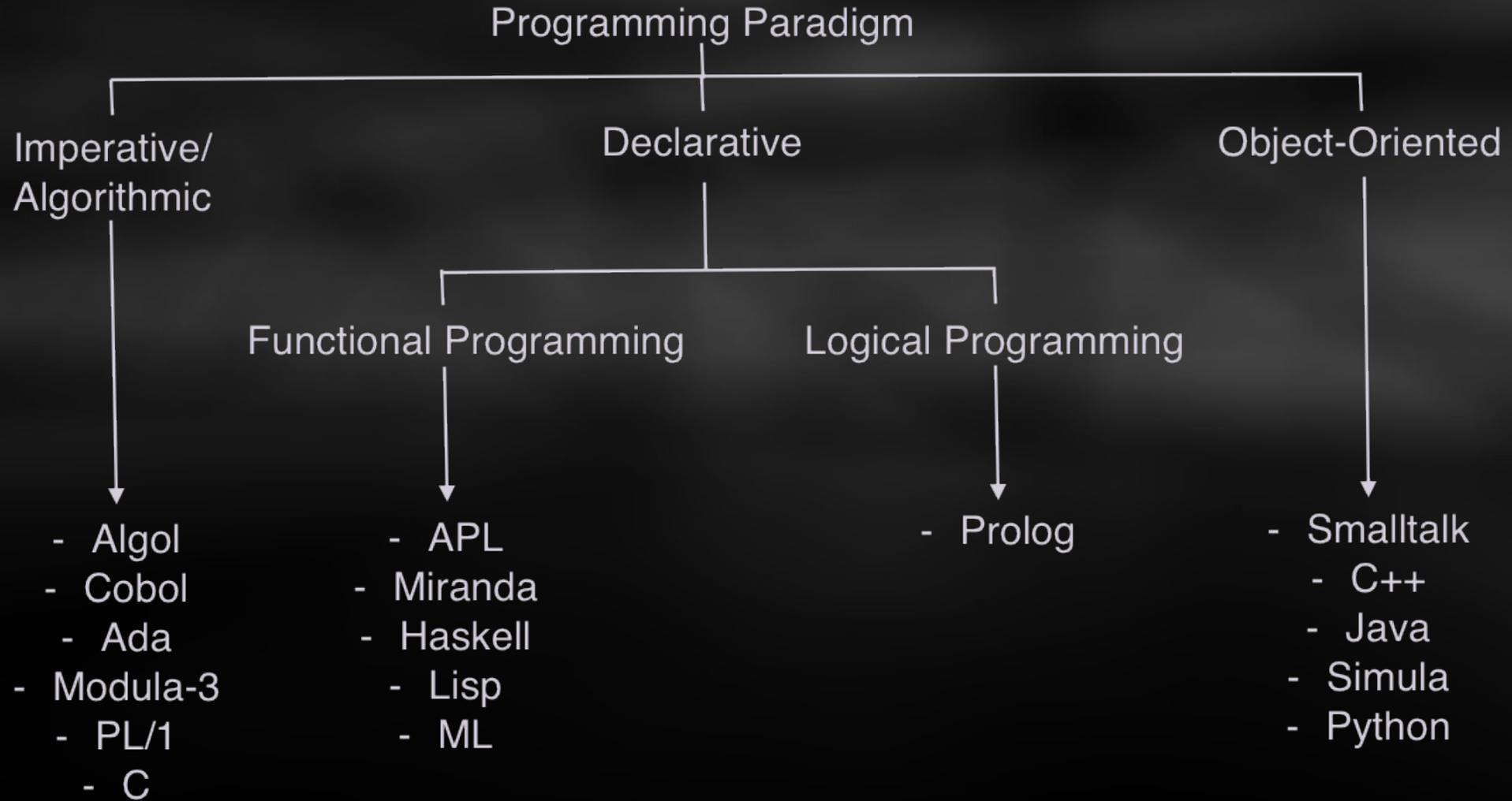
"More is not better (or worse) than less, just different."

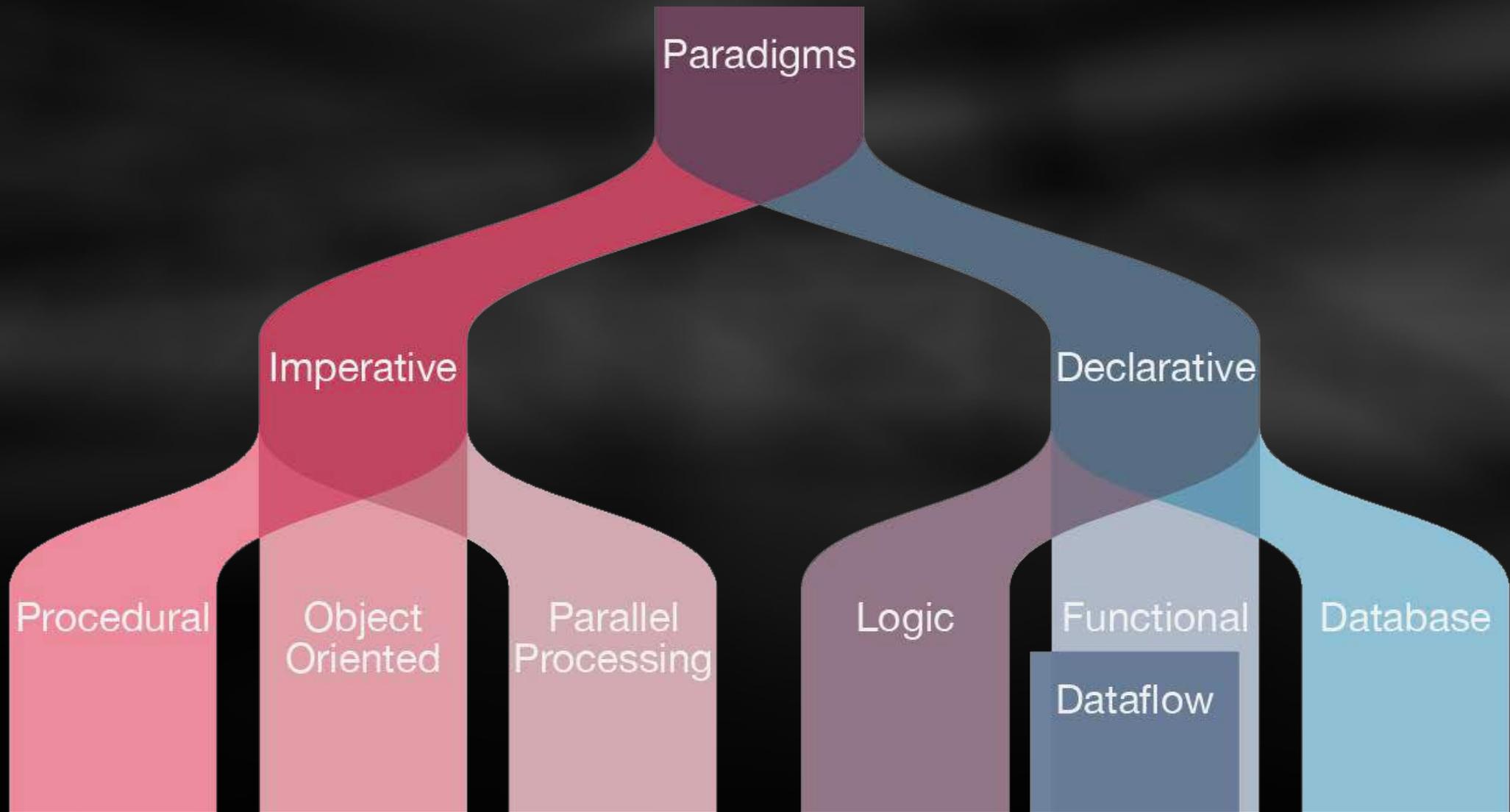
v1.08 © 2008 by Peter Van Roy

Data structures only  
Turing equivalent

Observable nondeterminism?  Yes  No







编程范式	描述	主要的特性	相关的编程语言
<b>Imperative</b> 命令式	使用流程化的语句和过程直接控制程序的运行和数据状态。	直接赋值 常用的数据结构 全局变量、局部变量 Goto语句 顺序化数据的操作和迭代 以功能为主的模块化	C, C++, Java, PHP, Python, Ruby
<b>Functional</b> 函数式	通过数学函数表达式的方式来避免状态和可变的数据。	代码公式化 Lambda 表达式 函数的包装和嵌套（高阶函数、Pipeline、Currying、Map/Reduce/Filter） 递归（尾递归） 无数据共享或依赖 无副作用（并行、重构、组合）	C++, Clojure, Coffeescript, Elixir, Erlang, F#, Haskell, Lisp, Python, Ruby, Scala, SequenceL, SML
<b>Object-Oriented</b> 面向对象	把一组字段和作用在其上面的方法抽象成一个个对象。	对象封装 消息传递 隐藏细节 数据和接口抽象 多态 继承 对象的序列化和反序列	Common Lisp, C++, C#, Eiffel, Java, PHP, Python, Ruby, Scala
<b>Declarative</b> 声明式	定义计算的逻辑而不是定义具体的流程控制。	4GLs, spreadsheets, report program generators	SQL, 正规表达式, CSS, Prolog, OWL, SPARQL