

PRG1 (5): 関数と状態

脇田建

2016.10.17

授業の後半に

小テストを実施します

Q1の復習

置換を用いたプログラムの理解

置換を用いたプログラムの理解

プログラムを詳しく眺める

```
def areaOfDisk :           //計算の方法の名前の宣言  
(Double) => Double       //受け取る値と結果の値が属する集合を宣言  
=  
(r) =>                    //入力をあらわす変数を宣言  
{3.14*r*r}               //計算方法を宣言  
  
areaOfDisk(5)              //areaOfDisk の変数 r を 5 に置換して計算せよ
```


名前参照の意味

```
val radius = 5
```

```
3.14 * radius * radius
```

名前参照の意味

```
val radius = 5
```

val 名前 = 定義

参照された名前は
定義で置換

```
3.14 * radius * radius
```

※ radius: 半径

名前参照の意味

```
val radius = 5
```

```
3.14 * radius * radius
```

```
3.14 * 5 * radius
```

名前参照の意味

```
val radius = 5
```

```
3.14 * radius * radius
```

```
3.14 * 5 * radius
```

参照された名前は
定義で置換

名前参照の意味

```
val radius = 5
```

```
3.14 * radius * radius
```

```
3.14 * 5 * radius
```

```
3.14 * 5 * 5
```

名前参照の意味

```
val radius = 5
```

```
3.14 * radius * radius
```

```
3.14 * 5 * radius
```

```
3.14 * 5 * 5
```

式の要素がすべて値に
になったら計算

名前参照の意味

```
val radius = 5
```

```
3.14 * radius * radius
```

```
3.14 * 5 * radius
```

```
3.14 * 5 * 5
```

```
78.5
```

式変形で値が得られたら
評価の停止

まとめ：名前参照の意味

- ❖ (部分) 式が名前を含んでいる限り
 - ❖ 名前を定義で置き換え: `val 名前 = 定義`
- ❖ (すでに (部分) 式は名前を含まないので) 評価

関数呼び出しの意味

```
def areaOfDisk(radius: Double): Double = {  
    3.14 * radius * radius  
}  
  
areaOfDisk(5)
```

関数呼び出しの意味

関数呼び出し：

(1) 局所環境の構成

{ ... } のなかで作業

(2) 引数の受け渡し

実引数の値 → 仮引数

(3) 関数定義の置換

関数呼出 → 関数定義

```
def areaOfDisk(radius: Double): Double = {  
    3.14 * radius * radius  
}  
  
areaOfDisk(5)
```


関数呼び出しの意味

関数呼び出し：

(1) 局所環境の構成

{ ... } のなかで作業

(2) 引数の受け渡し

実引数の値 → 仮引数

(3) 関数定義の置換

関数呼出 → 関数定義

```
def areaOfDisk(radius: Double): Double = {  
    3.14 * radius * radius  
}
```

```
areaOfDisk(5)
```

仮引数

実引数

関数呼び出しの意味

関数呼び出し：

(1) 局所環境の構成

{ ... } のなかで作業

(2) 引数の受け渡し

実引数の値 → 仮引数

(3) 関数定義の置換

関数呼出 → 関数定義

```
def areaOfDisk(radius: Double): Double = {  
    3.14 * radius * radius  
}  
  
areaOfDisk(5)
```


関数呼び出しの意味

関数呼び出し：

(1) 局所環境の構成

{ ... } のなかで作業

(2) 引数の受け渡し

実引数の値 → 仮引数

(3) 関数定義の置換

関数呼出 → 関数定義

```
def areaOfDisk(radius: Double): Double = {  
    3.14 * radius * radius  
}
```

areaOfDisk(5)

```
{  
    areaOfDisk(5)  
}
```

関数呼び出しの意味

関数呼び出し：

(1) 局所環境の構成

{ ... } のなかで作業

(2) 引数の受け渡し

実引数の値 → 仮引数

(3) 関数定義の置換

関数呼出 → 関数定義

```
def areaOfDisk(radius: Double): Double = {  
    3.14 * radius * radius  
}  
  
areaOfDisk(5)  
  
{  
    areaOfDisk(5)  
}
```

「val 仮引数 = 実引数」という
宣言があったつもりになる

関数呼び出しの意味

関数呼び出し：

(1) 局所環境の構成

{ ... } のなかで作業

(2) 引数の受け渡し

実引数の値 → 仮引数

(3) 関数定義の置換

関数呼出 → 関数定義

```
def areaOfDisk(radius: Double): Double = {  
    3.14 * radius * radius  
}
```

```
areaOfDisk(5)
```

```
{  
    areaOfDisk(5)  
}
```

```
{  
    val radius = 5  
    areaOfDisk(5)  
}
```

関数呼び出しの意味

関数呼び出し：

(1) 局所環境の構成

{ ... } のなかで作業

(2) 引数の受け渡し

実引数の値 → 仮引数

(3) 関数定義の置換

関数呼出 → 関数定義

```
def areaOfDisk(radius: Double): Double = {  
    3.14 * radius * radius  
}
```

```
areaOfDisk(5)
```

```
{  
    areaOfDisk(5)  
}
```

```
{  
    val radius = 5  
    areaOfDisk(5)  
}
```


関数呼び出しの意味

関数呼び出し：

(1) 局所環境の構成

{ ... } のなかで作業

(2) 引数の受け渡し

実引数の値 → 仮引数

(3) 関数定義の置換

関数呼出 → 関数定義

```
def areaOfDisk(radius: Double): Double = {  
    3.14 * radius * radius  
}
```

```
areaOfDisk(5)
```

```
{  
    areaOfDisk(5)  
}
```

```
{  
    val radius = 5  
    areaOfDisk(5)  
}
```

```
{  
    val radius = 5  
    3.14 * radius * radius  
}
```

関数呼び出しの意味

関数呼び出し：

(4) 関数呼び出し式が消滅したら、変数参照を順次置換して、最後に評価

```
def areaOfDisk(radius: Double): Double = {  
  3.14 * radius * radius  
}
```

```
areaOfDisk(5)
```

```
{  
  areaOfDisk(5)  
}
```

```
{  
  val radius = 5  
  areaOfDisk(5)  
}
```

```
{  
  val radius = 5  
  3.14 * radius * radius  
}
```


クイズ: `fact(3)` の意味を計算せよ

関数呼び出し：

(1) 局所環境の構成

`{ ... }` のなかで作業

(2) 引数の受け渡し

実引数の値 → 仮引数

(3) 関数定義の置換

関数呼出 → 関数定義

要注意

```
def fact(n: Int): Int = {  
    if (n == 0) 1  
    else n * fact(n - 1)  
}
```

`fact(3)`

記憶する関数 (HtDP 34)

Memory for Functions

- ❖ ここまで学んできた関数は評価結果にのみ意味があったが、多くのプログラムはその動作についての記録を残すことに意義がある.
- ❖ 例：アドレス帳

実行履歴を記憶するプログラム

アドレス帳の例

```
println(f"あやの電話は ${lookup('aya', address_book)}")  
add('aya', "090-1234-5678")  
println(f"あやの電話は ${lookup('aya', address_book)}")
```


実行履歴を記憶するプログラム

信号機シミュレータの例

Next ボタンの働きについて考えてみよう

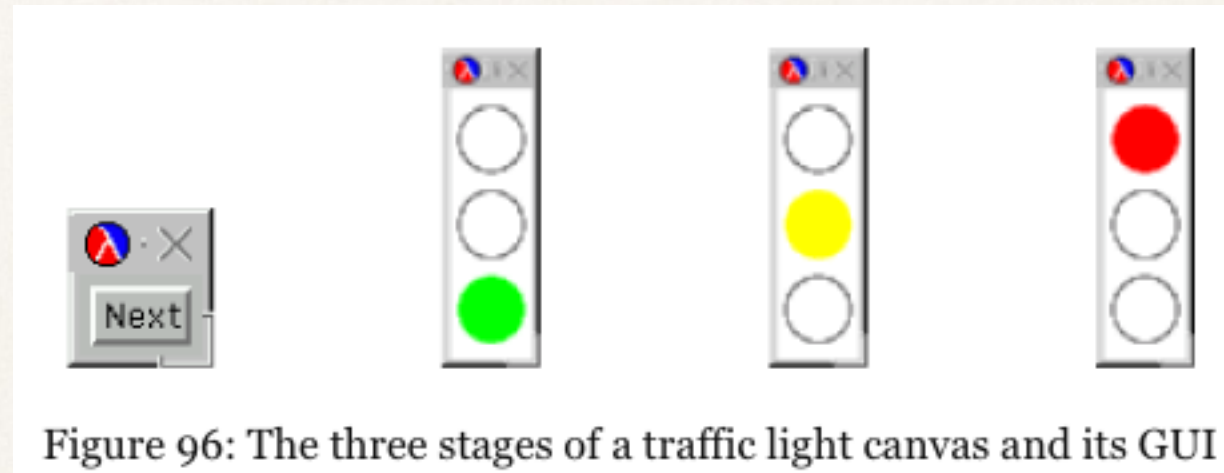


Figure 96: The three stages of a traffic light canvas and its GUI

実行履歴を記憶するプログラム

ハングマンの例

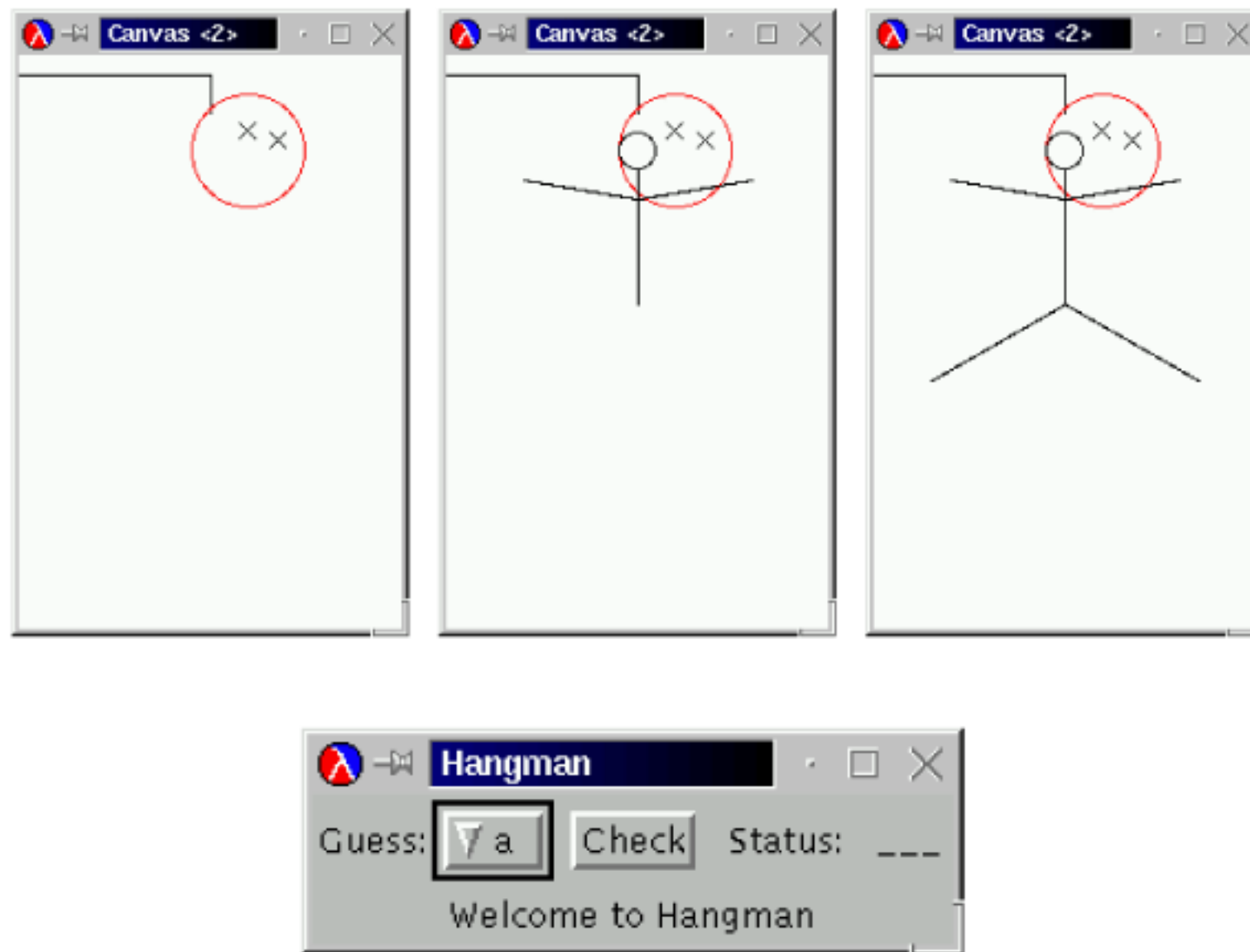


Figure 97: Three stages in the hangman game and its GUI

Check ボタンの働きについて考えてみよう

可変変数と代入

- ❖ **var** v = 初期値 // 可変変数（更新可能な / mutable）の宣言
 - ❖ v // 変数の参照は **val** 宣言された定数の参照と同様
 - ❖ $v = \dots$ // 代入式を用いて変数の値を更新できる。
代入式全体としての値は () でその型は Unit
- ❖ 混乱しがちな諸概念
 - ❖ 定数宣言: **val** v = 値 // 定数宣言は値に名前を与える
 - ❖ 同値関係: $x1 == y1$ // 同値関係は `==` であって `=` ではない


自習

- ❖ Scalaのインタプリタを起動し，
- ❖ `var x = 0` のように変数を宣言した上で，代入 `x = x + 1` によって，`x` に保存されている値が変化する様子を観察しなさい。
- ❖ つぎにこの現象を「置換を用いた理解」で説明できるか考えなさい。

代入を含むプログラムの理解

なにかがおかしい

```
{  
  val z = (x = x + 2)  
  x  
}  
  
{  
  val x = (3 = 3 + 2)  
  3  
}
```



代入を含むプログラムの理解

- ❖ 代入式には特殊な意味が与えられているため、単純な置換をしてはいけない
- ❖ 代入式の左辺は置換しない
- ❖ 代入式の意味
右辺の評価値を左辺の変数の新しい値とする
- ❖ 代入式の利用の前後でプログラムの状態が変化する。
代入式の評価以前の値で、代入式に続く式の評価をしない

代入の意味

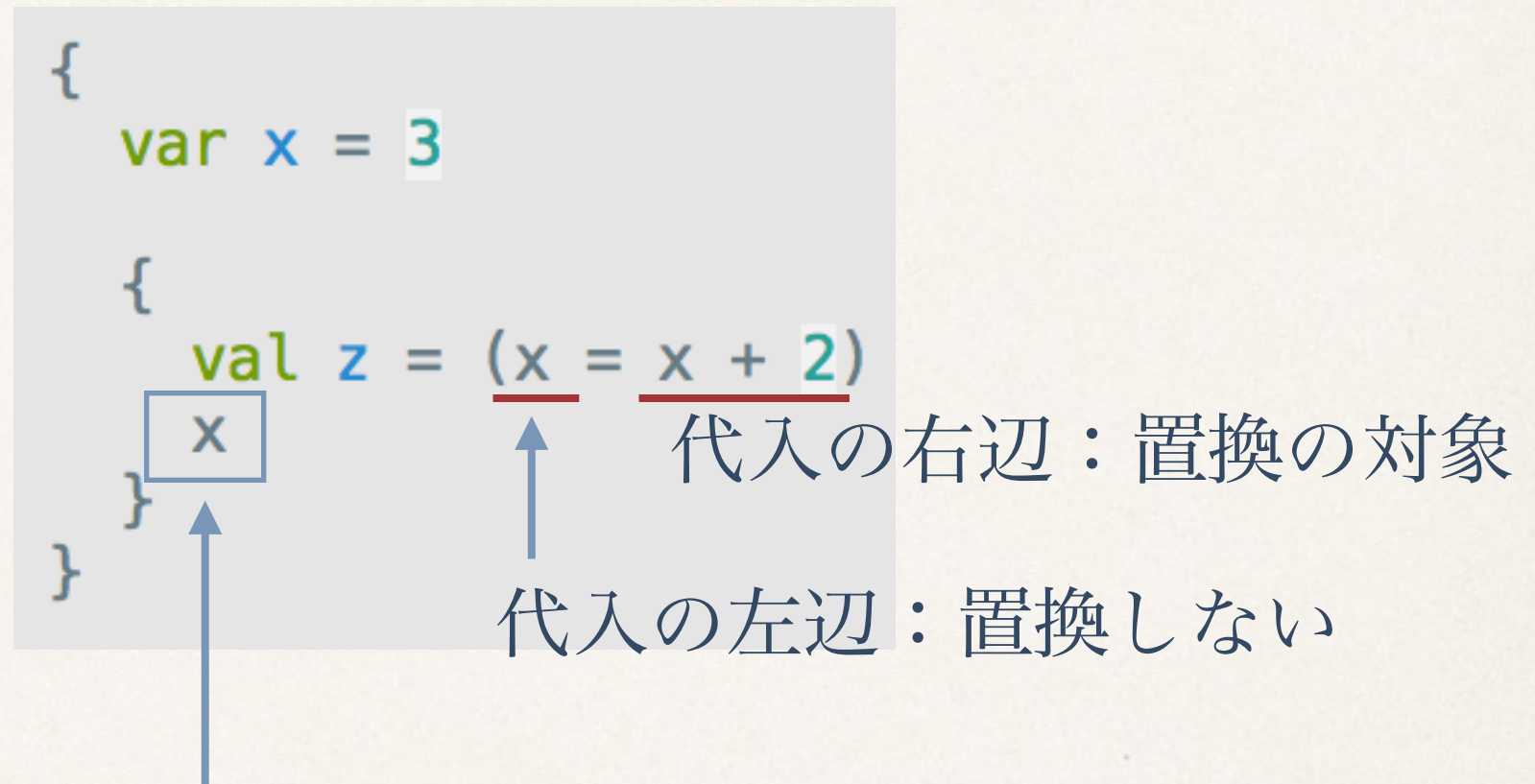
```
{  
  var x = 3  
  
  {  
    val z = (x = x + 2)  
    x  
  }  
}
```

↑ 代入の左辺

↑ 代入の右辺

- (1) 変数 z を初期化する前に代入式を評価する
- (2) どの変数を置換するかに注意

代入の意味



代入に続く式に対する置換は
代入の意味を反映してから行う

代入式の右辺について置換

```
{  
  var x = 3  
  
  {  
    val z = (x = x + 2)  
    x  
  }  
}  
  
{  
  var x = 3  
  
  {  
    val z = (x = 3 + 2)  
    x  
  }  
}
```

代入の右辺から変数が消えたら評価

```
{  
  var x = 3  
  
  {  
    val z = (x = 3 + 2)  
    x  
  }  
}  
  
{  
  var x = 3  
  
  {  
    val z = (x = 5)  
    x  
  }  
}
```


右辺の評価値で変数の値を更新

```
{  
  var x = 3  
  
  {  
    val z = (x = 5)  
    x  
  }  
}  
  
{  
  var x = 5  
  
  {  
    val z = ()  
    x  
  }  
}
```

(1) 変数の値を更新

(2) 代入式の値は (): Unit

x の値を通常どおりに置換

```
{  
  var x = 5  
  
  {  
    val z = ()  
    x  
  }  
}  
  
{  
  var x = 5  
  
  {  
    val z = ()  
    5  
  }  
}
```


x の値を通常どおりに置換

```
{  
  var x = 5  
  
  {  
    val z = ()  
    5  
  }  
}  
  
{  
  var x = 5  
  
  5  
}
```

{ ... } の意味は、この
中身のうち、一番最後
の要素の値

見た目、ほぼ同じだが...

1 という関数に `{ ... }` という内容の関数を

引数として渡す意味に解釈. 以下と同義

`1({ ... })`. 1は関数ではないので, 当然, エラー.

空行がはいると定数定義に続くブロックと解釈.

空行が重要な意味を持つことがある

❖ `val y = 1`
`{ ... }`

❖ `val y = 1`

`{ ... }`