R – Logics and Automation

Lecture 7

- Break statement
- While loop
- Equivalence of “for” loop and “while” loop

Looping

Looping allows to run the command many times. For example, if one needs to print a sentence, saying “I like Prince George”, it can be simply done by the command

print(“I like Prince George”)

“while loop” Loop in R

Similar to “if statement”, “while loop” sets the condition for statement. An essential difference between “if loop” and “while loop” is that the former always sets condition explicitly which is not related to statement at all, whereas the latter often sets the condition which is in turn determined by the statement. For example, we need to determine a number whose factorial is close to 1000 as much as possible, namely,

1*2*3*4*….*number <=1000

Syntax of while loop

```
while (condition)
{
  statement
}
```

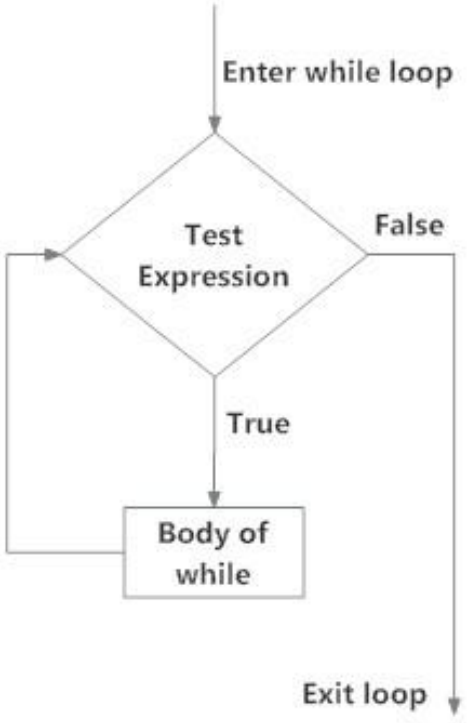
Here, condition is evaluated and the body of the loop is entered if the result is TRUE.

The statements inside the loop are executed and the flow returns to evaluate the “condition” again.

This is repeated each time until “condition” evaluates to FALSE, in which case, the loop exits.

Example of while Loop

```
i <- 1
while (i < 6) {
  print(i)
  i = i+1
}
```



In the above example, i is initially initialized to 1. Here, the condition is i < 6 which evaluates to TRUE since 1 is less than 6. So, the body of the loop is entered and i is printed and incremented.

Incrementing i is important as this will eventually meet the exit condition. Failing to do so will result into an infinite loop. In the next iteration, the value of i is 2 and the loop continues. This will continue until i takes the value 6. The condition 6 < 6 will give FALSE and the while loop finally exits.

“while” Loop in R

Find the number whose factorial is not greater than 1000

```
num<-1
sum<-1
while(sum<1000) {
  num<-num+1
  sum=sum*num
  }
```

Here, we try to determine the number whose factorial is up to 1000. We use while loop to iterate until the factorial is less than 1000. On each iteration, we multiply the number “num” to sum, which gives the total sum (factorial) in the end.

“while” Loop in R

You find the answer “num” is 7, 7! = 5040 which is much larger than 1000. The reason is while num=6, 6! = 720 smaller than 1000, the statement is still executed, i.e., num=num+1. So we should modify this code.. There are basically two ways, one is to use “if statement”, the other way is to change condition, i.e.,

```
num<-1
sum<-1
  while(sum*num<1000) {
num<-num+1
sum=sum*num
 }
```

“while” Loop in R

Example: Using “while loop” to produce a vector [1, 2, 3,…10]

```
storage<-c();
 x<-1
 while(x<10){
 storage<-c(storage,x)
 x<-x+1
 }
```

Example: double the vector of x=[1,2,3] five times, i.e. x1=2*x; x2=2*x1….

```
Counter<-1
x<-c(1,2,3)
While(counter<5){
x<-x*2;
print(x)
counter<-counter+1
}
```

“while” Loop in R

“while” loop can make infinite iteration so be careful in setting the condition for while loop. For example

```
x<-1
 while(x<2) {
 print(x)
}
```

Nested “while” loop

Example: there are more than one while loop used

```
i<-1
j<-1
while(i<=5){
while(j<=5){
print(c(i,j))
j<-j+1
}
i<-i+1
}
```

Example 4.6 “while” Loop

List all Fibonacci numbers (i.e. each number is the sum of the two preceding ones, starting from 0 and 1) less than 300. We do not know beforehand how long this list is. We do not know how to stop the for loop but a while loop seems perfect option for this example.

```
Fib1<-1
Fib2<-1
Fibonacii<-c(Fib1,Fib2)
While(Fib2<300){
 Fibonacii<-c(Finonacii,Fib2)
  oldFib2<-Fib2
   Fib2<-Fib1+Fib2
Fib1<-oldFib2
}
```

Using Next

Suppose you need to print all uneven numbers between 1 and 10 but even numbers should not be printed. In that case, your loop would look like this:

```
for (i in 1:10) {
  if (i %% 2==0){
    next
  }
print(i)
}
```

Example 4.1 (text book)

When i is between 1 and 10, the for loop is executed. In the for loop, we need to check if the value of i is odd or even.

If the value of i has a remainder of zero when divided by 2 (that's why we use the modulus operand %%), if statement is not executed. In case the remainder is nonzero, if statement evaluates to TRUE and "next" statement executes which move the loop back to the i in 1:10 condition thereby ignoring the the statement that follows.

Using Break

“break” is used inside a loop to stop the interactions and flow the control outside of the loop.

```
for(i in 1:5){print(i)}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
for(i in 1:5){if(i==3){break};print(i)}
[1] 1
[1] 2
```

“Break” Examples

```
x <- 1:5
for (val in x) {
 if (val == 3){
 break
}
print(val)
 }
```

```
x<-0
while (x < 10) {
 x <- x + 4
print (x)
 if (x = 8) {
 break
 }
}
```

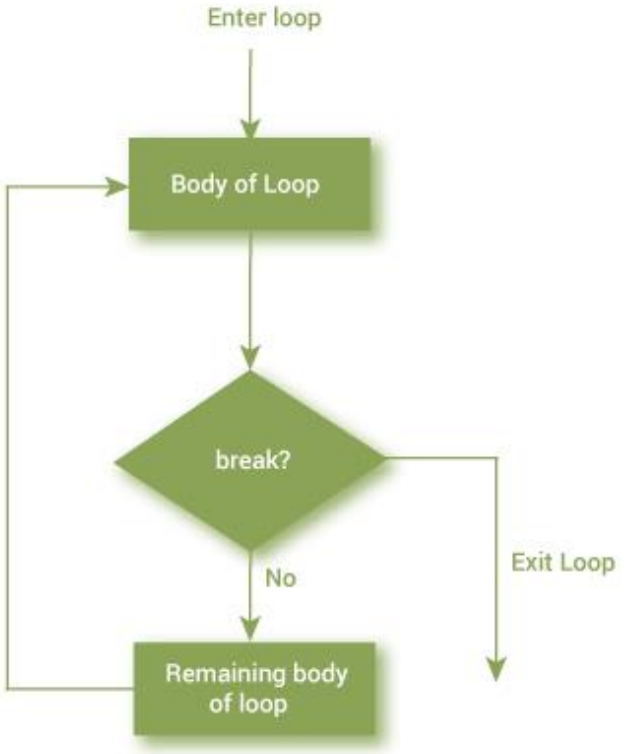
Using Repeat loop

The repeat() loop is to do repetition without condition. In other words, it is an infinite repetition. The syntax is like

```
repeat {
statement
}
```

Example

```
repeat{print("I like PG")}
```



Using Repeat

To end the loop, one must use condition and "break", with the syntax

```
repeat {
statement
if(condition){break}
}
```

Repeat Example

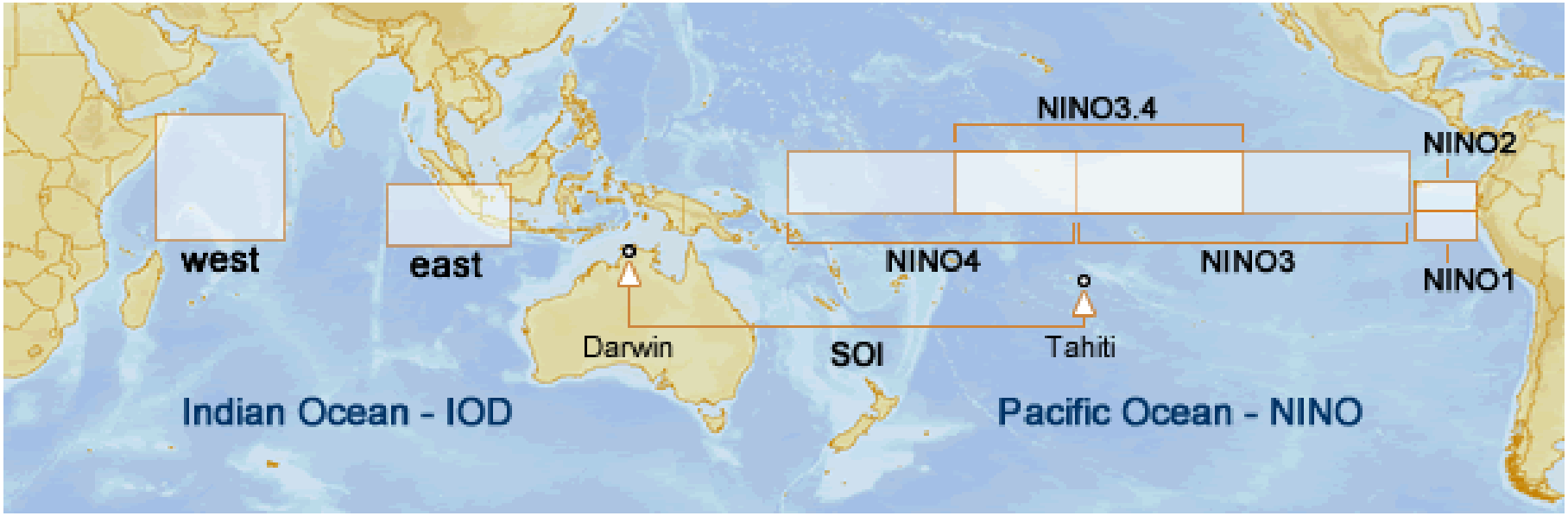
```
x <- 1
repeat {
 print(x)
 x = x+1
 if (x == 10){
break }
}
```

we have used a condition to check and exit the loop when x is equal to 10.

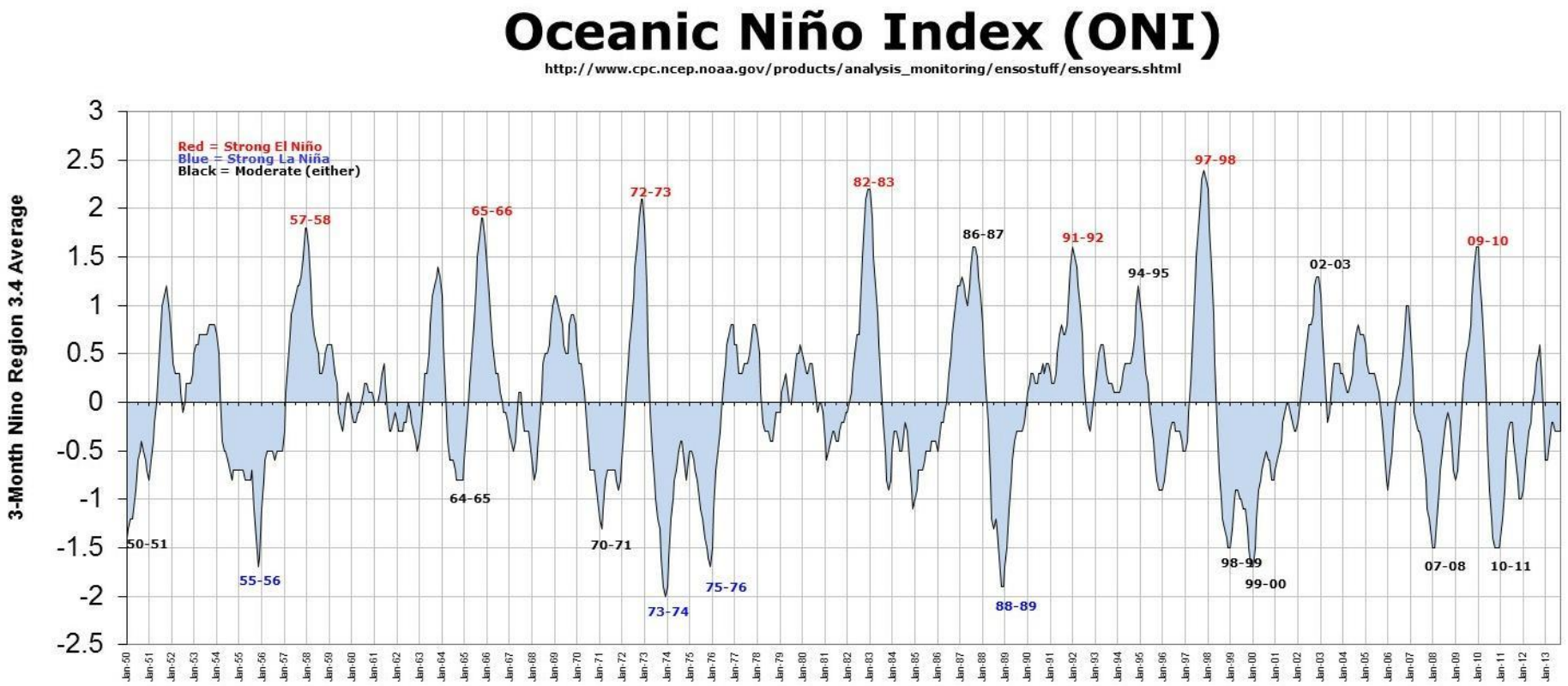
mortgage.R

```
# set up
month <- 0       # count the number of months
balance <- 300000 # initial mortgage balance
payments <- 1600  # monthly payments
interest <- 0.03  # 5% interest rate per year
total.paid <- 0   # track what you've paid the bank
# convert annual interest to a monthly multiplier
monthly.multiplier <- (1+interest) ^ (1/12)
# keep looping until the loan is paid off...
while ( balance > 0 ) {
  # do the calculations for this month
  month <- month + 1  # one more month
  balance <- balance * monthly.multiplier  # add the interest
  balance <- balance - payments  # make the payments
  total.paid <- total.paid + payments # track the total paid
  # print the results on screen
  cat( "month", month, ": balance", round(balance), "\n")
}
# print the total payments at the end
cat("total payments made", total.paid, "\n" )
```

SST Data


NINO3.4
NINO2
west
east
NINO4
NINO3
NINO1
Darwin
SOI
Tahiti
Indian Ocean - IOD
Pacific Ocean - NINO

Composite Analysis



The Oceanic Niño Index (ONI) has become the de-facto standard that NOAA uses for identifying warm and cool events in the tropical Pacific. It is the running 3-month mean SST anomaly for the Nino 3.4 region (i.e., 5°N-5°S, 120°-170°W) e.g. warm events are defined as seasons above the 0.5° anomaly and strong warm events as > 1.5°.

Lab Exercise

1. Donwload the data http://web.unbc.ca/~islam/ENSC250/ONI_1950-2019.txt
2. Load the text file into R.

3. Plot the real and anomaly seasonal data.

4. Apply composite analysis i.e. select warm and cold months and save them in different arrays.
   - If SST anomaly is > 0.5 Warm
   - If SST anomaly is > 1.5 Strong Warm
   - If SST anomaly is > -0.5 Cold
   - If SST anomaly is > -1.5 Strong Cold
5. Take the mean of arrays for composites.