Abhishek Murthy
21BDS0064
Fall Sem 2024-2025
DA -1.1
Exploratory Data Analysis Lab
27-07-2024

Branching:

1. Check age group

```
age <- 25
if (age <= 19) {
  print("Teenager")
} else if (age >= 20 & age <= 59) {
  print("Adult")
} else {
  print("Senior Citizen")
}
```

```
# 1. Check age group
age <- 25
if (age <= 19) {
  print("Teenager")
} else if (age >= 20 & age <= 59) {
  print("Adult")
} else {
  print("Senior Citizen")
}
```

Output:

```
> # 21BDS0064
> age <- 25
> if (age <= 19) {
+   print("Teenager")
+ } else if (age >= 20 & age <= 59) {
+   print("Adult")
+ } else {
+   print("Senior Citizen")
+ }
[1] "Adult"
```

2. Show the difference between two strings

```
name1 = "abhi"
name2 = "Abhi"
if(name1 == name2){
  print("Same two strings")
} else{
  print("Different strings")
}
```

```
# 2. Show the difference between two strings
name1 = "abhi"
name2 = "Abhi"
if(name1 == name2){
  print("Same two strings")
} else{
  print("Different strings")
}
```

Output:

```
> # 21BDS0064
> name1 = "abhi"
> name2 = "Abhi"
> if(name1 == name2){
+   print("Same two strings")
+ } else{
+   print("Different strings")
+ }
[1] "Different strings"
```

3. Salary brackets

```
salary = 3000000
if(salary < 700000){
  print("Bracket 1")
} else if(salary > 700000 & salary < 1500000){
  print("Bracket 2")
} else{
  print("Bracket 3")
}
```

```
# 3. Salary brackets
salary = 3000000
if(salary < 700000){
  print("Bracket 1")
} else if(salary > 700000 & salary < 1500000){
  print("Bracket 2")
} else{
  print("Bracket 3")
}
```

Output:

```
> # 21BDS0064
> salary = 3000000
> if(salary < 700000){
+   print("Bracket 1")
+ } else if(salary > 700000 & salary < 1500000){
+   print("Bracket 2")
+ } else{
+   print("Bracket 3")
+ }
[1] "Bracket 3"
```

4. Grade checker

```
score <- 85
if (score >= 90) {
  print("S Grade")
} else if (score >= 80 & score < 90) {
  print("A Grade")
} else if (score >= 70 & score < 80) {
  print("B Grade")
} else if (score >= 60 & score < 70) {
  print("C Grade")
} else if (score >= 50 & score < 60) {
  print("D Grade")
} else if (score >= 40 & score < 50) {
  print("E Grade")
}else {
  print("Fail")
}
```

```
# 4. Grade checker
score <- 85
if (score >= 90) {
  print("S Grade")
} else if (score >= 80 & score < 90) {
  print("A Grade")
} else if (score >= 70 & score < 80) {
  print("B Grade")
} else if (score >= 60 & score < 70) {
  print("C Grade")
} else if (score >= 50 & score < 60) {
  print("D Grade")
} else if (score >= 40 & score < 50) {
  print("E Grade")
}else {
  print("Fail")
}
```

Output:

```
> # 21BDS0064
> score <- 85
> if (score >= 90) {
+   print("S Grade")
+ } else if (score >= 80 & score < 90) {
+   print("A Grade")
+ } else if (score >= 70 & score < 80) {
+   print("B Grade")
+ } else if (score >= 60 & score < 70) {
+   print("C Grade")
+ } else if (score >= 50 & score < 60) {
+   print("D Grade")
+ } else if (score >= 40 & score < 50) {
+   print("E Grade")
+ }else {
+   print("Fail")
+ }
[1] "A Grade"
```

5. Switch method for basic arithmetic operations

```
operation <- "+"
a <- 5
b <- 3
result <- switch(operation,
            "+" = a + b,
            "-" = a - b,
            "*" = a * b,
            "/" = a / b)
print(result)
```

```
# 5. Switch method for basic arithmetic operations
operation <- "+"
a <- 5
b <- 3
result <- switch(operation,
                 "+" = a + b,
                 "-" = a - b,
                 "*" = a * b,
                 "/" = a / b)
print(result)
```

Output:

```
> # 21BDS0064
> operation <- "+"
> a <- 5
> b <- 3
> result <- switch(operation,
+                  "+" = a + b,
+                  "-" = a - b,
+                  "*" = a * b,
+                  "/" = a / b)
> print(result)
[1] 8
```

6. Check the type of triangle

```
triangleType <- function(a, b, c) {
 if (a == b && b == c) {
  type <- "Equilateral"
 } else if (a == b || b == c || a == c) {
  type <- "Isosceles"
 } else {
  type <- "Scalene"
 }
 return(type)
}
triangleType(3, 3, 3)
triangleType(3, 3, 2)
triangleType(3, 4, 5)
```

```
# 6. Type of triangle
triangleType <- function(a, b, c) {
  if (a == b && b == c) {
    type <- "Equilateral"
  } else if (a == b || b == c || a == c) {
    type <- "Isosceles"
  } else {
    type <- "Scalene"
  }
  return(type)
}

triangleType(3, 3, 3)
triangleType(3, 3, 2)
triangleType(3, 4, 5)
```

Output:

```
> # 21BDS0064
> triangleType <- function(a, b, c) {
+   if (a == b && b == c) {
+     type <- "Equilateral"
+   } else if (a == b || b == c || a == c) {
+     type <- "Isosceles"
+   } else {
+     type <- "Scalene"
+   }
+   return(type)
+ }
>
> triangleType(3, 3, 3)
[1] "Equilateral"
> triangleType(3, 3, 2)
[1] "Isosceles"
> triangleType(3, 4, 5)
[1] "Scalene"
```

Looping:

1. Square of a number that is less than 900

```
n = 0
square = 0
while(square <= 900){
 n = n+1
 square = n^2
}
n
square
```

```
# 1. Square of a number that is less than 900
n = 0
square = 0
while(square <= 900){
  n = n+1
  square = n^2
}
n
square
```

Output:

```
> # 21BDS0064
> n = 0
> square = 0
> while(square <= 900){
+   n = n+1
+   square = n^2
+ }
> n
[1] 31
> square
[1] 961
```

2. Simple Interest

```
savings <- 1000
interest_rate <- 0.05
years <- 0

while (savings < 2000) {
 savings <- savings * (1 + interest_rate)
 years <- years + 1
}
print(paste("It takes", years, "years to double the savings."))
```

```
# 2. Simple Interest
savings <- 1000
interest_rate <- 0.05
years <- 0

while (savings < 2000) {
  savings <- savings * (1 + interest_rate)
  years <- years + 1
}
print(paste("It takes", years, "years to double the savings."))
```

Output:

```
> # 21BDS0064
> savings <- 1000
> interest_rate <- 0.05
> years <- 0
>
> while (savings < 2000) {
+   savings <- savings * (1 + interest_rate)
+   years <- years + 1
+ }
> print(paste("It takes", years, "years to double the savings."))
[1] "It takes 15 years to double the savings."
```

3. Compound Interest

```r
principal <- 15000
annual_contribution <- 3500
interest_rate <- 0.1
years <- 12

for(yr in 1:years) {
  principal <- principal * (1 + interest_rate) + annual_contribution
  print(paste(yr, "th year", ":", round(principal, 4)))
}
```

```r
# 3. Compound Interest
principal <- 15000
annual_contribution <- 3500
interest_rate <- 0.1
years <- 12

for(yr in 1:years) {
  principal <- principal * (1 + interest_rate) + annual_contribution
  print(paste(yr, "th year", ":", round(principal, 4)))
}
```

Output:

```
> # 21BDS0064
> principal <- 15000
> annual_contribution <- 3500
> interest_rate <- 0.1
> years <- 12
>
> for(yr in 1:years) {
+   principal <- principal * (1 + interest_rate) + annual_contribution
+   print(paste(yr, "th year", ":", round(principal, 4)))
+ }
[1] "1 th year : 20000"
[1] "2 th year : 25500"
[1] "3 th year : 31550"
[1] "4 th year : 38205"
[1] "5 th year : 45525.5"
[1] "6 th year : 53578.05"
[1] "7 th year : 62435.855"
[1] "8 th year : 72179.4405"
[1] "9 th year : 82897.3846"
[1] "10 th year : 94687.123"
[1] "11 th year : 107655.8353"
[1] "12 th year : 121921.4188"
```

4. House robber where a robber should be able pick the most amount of coins

```r
rob <- function(nums) {
 if (length(nums) == 0) return(0)
 if (length(nums) == 1) return(nums[1])
 second <- nums[1]
 prev <- nums[2]
 for (i in 3:length(nums)) {
  curr <- max(nums[i] + second, prev)
  second <- prev
  prev <- curr
 }
 return(prev)
}
nums <- c(2, 7, 9, 3, 1)
rob(nums)
```

```r
# 4. House robber where a robber should be able pick the most amount of coins
rob <- function(nums) {
  if (length(nums) == 0) return(0)
  if (length(nums) == 1) return(nums[1])

  second <- nums[1]
  prev <- nums[2]

  for (i in 3:length(nums)) {
    curr <- max(nums[i] + second, prev)
    second <- prev
    prev <- curr
  }
  return(prev)
}
nums <- c(2, 7, 9, 3, 1)
rob(nums)
```

Output:

```
> # 21BDS0064
> rob <- function(nums) {
+   if (length(nums) == 0) return(0)
+   if (length(nums) == 1) return(nums[1])
+
+   second <- nums[1]
+   prev <- nums[2]
+
+   for (i in 3:length(nums)) {
+     curr <- max(nums[i] + second, prev)
+     second <- prev
+     prev <- curr
+   }
+
+   return(prev)
+ }
> nums <- c(2, 7, 9, 3, 1)
> rob(nums)
[1] 12
```

5. Majority element in an element where an element appears more than n/2

```
majorityElement <- function(nums) {
 count <- 0
 candidate <- NULL
 for (i in nums) {
  if (count == 0) {
   candidate <- i
  }

  if (i == candidate) {
   count <- count + 1
  } else {
   count <- count - 1
  }
 }
 return(candidate)
}

nums <- c(3, 2, 3, 1, 3)
majorityElement(nums)
```

```
# 5. Majority element in an element where an element appears more than n/2
majorityElement <- function(nums) {
  count <- 0
  candidate <- NULL
  for (i in nums) {
    if (count == 0) {
      candidate <- i
    }

    if (i == candidate) {
      count <- count + 1
    } else {
      count <- count - 1
    }
  }
  return(candidate)
}

nums <- c(3, 2, 3, 1, 3)
majorityElement(nums)
```

Output:

```
> # 21BDS0064
> majorityElement <- function(nums) {
+   count <- 0
+   candidate <- NULL
+   for (i in nums) {
+     if (count == 0) {
+       candidate <- i
+     }
+
+     if (i == candidate) {
+       count <- count + 1
+     } else {
+       count <- count - 1
+     }
+   }
+   return(candidate)
+ }
>
> nums <- c(3, 2, 3, 1, 3)
> majorityElement(nums)
[1] 3
```

6. Monitoring the uptime of a server, you log whether the server is up (1) or down (0) each minute in an array

```
findMaxConsecutiveOnes <- function(server){
 count <- 0;
 maxCount <- 0;
 for(r in server){
  if(r == 1){
   count = count+1;
  }
  else{
   count = 0;
  }
  maxCount = max(maxCount, count);
 }
 return(maxCount)
}
server <- c(1,1,0,1,1,1)
findMaxConsecutiveOnes(server)
```

```
# 6. Monitoring the uptime of a server.
# You log whether the server is up (1) or down (0) each minute in an array
findMaxConsecutiveOnes <- function(server){
  count <- 0;
  maxCount <- 0;
  for(r in server){
    if(r == 1){
      count = count+1;
    }
    else{
      count = 0;
    }
    maxCount = max(maxCount, count);
  }
  return(maxCount)
}

server <- c(1,1,0,1,1,1)
findMaxConsecutiveOnes(server)
```

Output:

```
> # 21BDS0064
> findMaxConsecutiveOnes <- function(server){
+   count <- 0;
+   maxCount <- 0;
+   for(r in server){
+     if(r == 1){
+       count = count+1;
+     }
+     else{
+       count = 0;
+     }
+     maxCount = max(maxCount, count);
+   }
+   return(maxCount)
+ }
>
> server <- c(1,1,0,1,1,1)
> findMaxConsecutiveOnes(server)
[1] 3
```