4. Hafta - Ruby Ders İçeriği (Ruby week1)

Dr. Öğr. Üyesi İbrahim KÖK

- Temel Kavramlar-Basics Concepts
- Değişkenler ve Veri Türleri-Data types
- Atamalar-Assignments
- Karar ve Kontrol durumları- Control statements
- Döngüler- Loops

Ruby Basics

Kurulum linkleri

- Linux ve diğer https://www.ruby-lang.org/en/documentation/installation/
- anaconda için https://anaconda.org/conda-forge/ruby
- windows için https://rubyinstaller.org/

Yorum satırları- Comments Genelde line-comment yani satır bazlı, ve blockcomment yani kod bloğu bazlı yorum yapma şekilleri var.

- Tek satır Comment için # işareti kullanılıyor.
- Çok satır için =begin ve =end

In [1]:
```
# This entire line is a comment
```

In [2]:
```
=begin
çoksatırlı
yorum
yazıyoruz
=end
```

Rezerve edilmiş kelimeler

Her dilde olduğu gibi Ruby'de de daha önceden rezerve edilmiş bazı kelimeler bulunmaktadır. Bu kelimeleri değişken ya da fonksiyon/method adı olarak kullanamıyoruz. Bu kelimeler Ruby komutları ve özel durumlar için rezerve edilmiş.

https://www.tutorialspoint.com/ruby/ruby_syntax.htm

Input -Output

- ** Çıktı için puts veya print
  - Puts ---> add line break at the end of output- otomatik olarak alt satıra geçer.
  - print ---> doesnt create line break at the end of output-geçmez
- **Girdi** Kullanıcıdan girdi almak için **gets**
  - sayi=gets
  - sayi=gets.chomp
  - sayi=gets.chomp.to_i

In [9]:
```
#puts  çıktı sonunda yeni satıra geçer
puts "Merhaba Pau"
puts "Merhaba CENG"
```

```
Merhaba Pau
Merhaba CENG
```

In [10]:
```
#puts  çıktı sonunda aynı satırdan devame eder
print "Merhaba Pau"
print "Merhaba CENG"
```

```
Merhaba PauMerhaba CENG
```

In [ ]:
```

```

Veri Türleri (Data Types)

- Object
- Fixnum They are normal numbers
- Bignum They are big numbers
- Float Decimal numbers
- Complex Imaginary numbers 4 + 3i
- Rational They are fractional numbers 9/4
- BigDecimal Precision decimal numbers 6.0

In [3]:
```
# Ruby Duck Typing şeklinde çalışır.
#Ördek Yazımı : "Ördek gibi yürüyorsa ve ördek gibi vaklıyorsa, o zaman bir ördek olmalıdı
#Yani atama yapmadan önce ne tür bir değer ataması yapacağımızı belirtmemize gerek yok.
o = Object.new
puts o.object_id
```

```
70202607850960
```

In [6]:
```
# o = 5 dediğimizde, Ruby "Hmmm, bu değer Fixnum türünde" diye değerlendirir.
o=5.0
o.class
```

Out[6]:
```
Float
```

In [7]:
```
#Variable, nesneye atanmis bir isaretcidir.Ruby'de her sey bir nesne yani Object.
#degiskenler nesnelere erişmek için aracıdır
#degisken isimleri KUCUK harfle veya _ alt cizgi ile baslamalidir.
#degisken isimleri case-sensitive dir.
#degisken isimleri için rezerve edilmis keyword leri kullanmayin.(i.e. puts or print)

  x1 = 7 # x1.class => Fixnum

  _x2  = "a string"            # _x2.class => String
   cF  = 4611686018427387903   # cF.class => Fixnum
   cB  = 4611686018427387904   # cB.class => Bignum

  x_3 = 2.3          # x.class => Float
```

```
  x4 = true          # x.class => TrueClass    --Mantıksal (Boolean) islem,

  x5 = false         # x.class => FalseClass   -- Mantıksal (Boolean) islem,

  x6 = nil           # x.class => NilClass   - Tanımı olmayan (Undefined) seylerin degeri.
  aBc=4

#it is an object represent nothing

puts x1,_x2,x_3,x4,x5,x6
```

```
7
a string
2.3
true
false
```

In [8]:

```
x1 = 7
puts x1.class # Integer
puts x1.is_a?(String) # false
puts x1.to_s.class # String
```

```
Fixnum
false
String
```

In [ ]:

```
# Atama islemlerinde önce esitligin sag tarafındaki islem gerceklestirilir. sonra atama ya
puts  v1=24+2
```

In [ ]:

```
# Atama islemlerinde önce esitligin sag tarafındaki islem gerceklestirilir. sonra atama ya
puts _v3=34
puts _v3+5
```

Assingments- Atamalar

In [11]:

```
#ayrı satırlarda atama işlemi
a=10
b=30
c=40
puts a
puts b
puts c
```

```
10
30
40
```

In [12]:

```
# tek satırda atama işlemleri
a,b,c,d=10,20,30,40
puts a,b,c,d
```

```
10
20
30
40
```

In [13]:

```
#swapping values between variables
```

```
a,b = c,d
puts a,b,c,d
```

```
30
40
30
40
```

Constants-Sabitler

In [ ]:
```
# Sabitler BUYUK harfle baslar
MY_NAME="ibrahim"
Surname="KÖK"
PI=3.14159

puts PI
```

In [ ]:
```
Surname="new surname"
```

In [ ]:
```
puts "My name is #{MY_NAME}"    # formatted output #{variable name}
```

In [ ]:
```
MY_NAME="gülsah"    #ruby de sabit değgstirmek istediğimiz de-önceden tanimlandigi uyaris
```

In [ ]:
```
puts "My name is #{MY_NAME}"
```

İşleçler- Ruby Operators

Basic Arithmetic operators are '+','-','*','/' and '%'.

In [14]:
```
puts("add operator 10 + 20")
 puts(10 + 20)
 puts("subtract operator 35 - 15")
 puts(35 - 15)
 puts("multiply operator 4 * 8")
 puts(4 * 8)
 puts("division operator 25 / 5")
 puts(25 / 5)
```

```
add operator 10 + 20
30
subtract operator 35 - 15
20
multiply operator 4 * 8
32
division operator 25 / 5
5
```

In [15]:
```
puts 9.0/2
puts 9/2.0
puts -9/2.0
puts("exponential operator")
puts(5 ** 2)
puts("modulo operator")
puts(25 % 4)
```

```
4.5
4.5
-4.5
exponential operator
25
modulo operator
1
```

Comparison Operators compare two operands.

== Equal operator .eql?, equal? != Not equal operator > left operand is greater than right operand < Right operand is greater than left operand >= Left operand is greater than or equal to right operand <= Right operand is greater than or equal to left operand <=> Combined comparison operator*** === Used to test equality within a when clause of a case statement.(1...10) === 5 returns true.

In [16]:
```
puts("Comparison operator")
puts(2 == 5)
puts (2.eql?5)
puts (2.equal?5)
puts(2 != 5)
puts(2 > 5)
puts(2 < 5)
puts(2 >= 5)
puts(2 <= 5)
```

```
Comparison operator
false
false
false
true
false
true
false
true
```

a<=>b Birleşik karşılaştırma operatörü -- Combined comparison operator

- if a < b then return -1
- if a = b then return 0
- if a > b then return 1
- if a and b are not comparable then return nil

In [19]:
```
a=(5 <=> 5)
puts a
```

```
0
```

In [20]:
```
#deimal/hex egerlere göre karşılaştırma
puts ("b" <=> "B")
```

```
1
```

Range operators create a range of successive values consisting of a start, end and range of values in between **

- **..** Range is **inclusive** of the last term 1..10 (1-10)
- **...** Range is **exclusive** of the last term 1...10 (1-9)

In [22]:
```
(1...10).to_a
```

Out[22]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [23]:
```
#If you need to, you can convert a range to a list using the   to_a    method. Python(5),
range1=(1..10).to_a
range2=(1...10).to_a
range_str=('ab'..'af').to_a

puts "#{range1}"
puts "#{range2}"
puts "#{range_str}"
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
["ab", "ac", "ad", "ae", "af"]
```

In [24]:
```
range3=(5...20).step(4).to_a
```

Out[24]:
```
[5, 9, 13, 17]
```

Convert methods

Tip degistirmek icin kullanılır.

- to_i , to_f , to_s , to_str , to_sym , to_r , to_c, to_a

In [ ]:
```
"merhaba".to_r   # => (0/1) # Rasyonel sayi
"0.2".to_r       # => (1/5) # Rasyonel sayi
"merhaba".to_c   # => (0+0i) # Kompleks sayi
"1234".to_c      # => (1234+0i)
#"merhaba".to_enum # => #<Enumerator: "merhaba":each> # Enumeratör'e cevirdi
```

In [ ]:
```
"merhaba".to_i   # => 0 # integer'a cevirdi
```

In [ ]:
```
"59".to_i   # => 0 # integer'a cevirdi
```

In [ ]:
```
"merhaba".to_f   # => 0.0 # float'a cevirdi
```

In [ ]:
```
"5".to_i         # => 5
```

In [ ]:
```
"1".to_f       # => 1.5
```

In [ ]:
```
5.to_s   # => "merhaba" # string
```

In [ ]:
```
"merhaba".to_str # => "merhaba" # string
```

Ranges implement methods that let you iterate over them and test their contents in a variety of ways

- include, min, max, reject

In [32]:
```
#to.a   array a çevirme- python daki list veri türüne denk geliyor
sayilar=(1..9).to_a
#sayilar.include?(15)   # belirtilen aralıkta 5 var mı?  cover, member, include

#sayilar.member?(15)
```

Out[32]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [34]:
```
puts "max değer #{sayilar.max}"
```

```
max değer 9
```

In [ ]:
```
#min max
mini=sayilar.min
maxi=sayilar.max
puts "minimum değer #{sayilar.min}"
puts "minimum değer #{maxi}"
```

In [35]:
```
#reject ***
kalan = sayilar.reject {|i| i < 5 }
puts "Kalan sayilar #{kalan}"
```

```
Kalan sayilar [5, 6, 7, 8, 9]
```

Logical operators

- && AND operator
- || OR operator

Object Methods

Everthing is an object in Ruby. String, integers anf floating point is an object. So they have methods such as;

```
* string ---> length, upcase, downcase, capitalize , center , chars , chomp
, chop , clear , count ,
* size , length , delete , ljust , rjust , reverse , upcase , downcase ,
swapcase ,
* reverse , index , hex , rindex , insert
```

In [36]:
```
puts "this is the rubyin Pau".length
```

```
22
```

In [37]:
```
puts "this is the ruby course in Pau".upcase
```

```
THIS IS THE RUBY COURSE IN PAU
```

In [38]:
```
puts "THIS IS THE RUBY COURSE IN PAU".downcase
```

```
this is the ruby course in pau
```

In [39]:
```
foo="Hello Pau"
foo.length
```

Out[39]: 9

In [ ]:
```
foo.upcase
```

In [40]:
```
"merhaba".ljust(20)
```

Out[40]: "merhaba   "

In [41]:
```
"merhaba".rjust(20, "*")
```

Out[41]: "************merhaba"

In [42]:
```
"merhaba".insert(0, "X")
```

Out[42]: "Xmerhaba"

In [43]:
```
"merhaba".insert(3, "A")
```

Out[43]: "merAhaba"

In [44]:
```
"merhaba dünya".reverse
```

Out[44]: "ayn\u00FCd abahrem"

In [45]:
```
"merhaba dünya".count("a")
```

Out[45]: 3

In [46]:
```
"Merhaba Dünya".delete("e")
```

Out[46]: "Mrhaba D\u00FCnya"

Kontrol durumları- Control statements

Döngüler- Loops

In [ ]:
```
#print 'Bir sayı girin: '
#sayi = gets.chomp.to_i # Kullanıcıdan değer alma
```

In [47]:
```
sayi=50
if sayi<10
    puts "sayı 10'dan küçük"
elsif sayi>=10 and sayi<=20
    puts "sayı 10 ile 20 arasında"
else
    puts "sayı 20'den büyük"
end
```

```
sayı 20'den büyük
```

In [48]:
```ruby
sayi=50
unless sayi>=20    #unless --> if in tersi anlamında yani if not şeklinde düşünülebilir.
    puts "sayı 20'den küçük"
else
    puts "sayı 20'den büyük veya eşit"
end
```

```
sayı 20'den küçük
```

In [56]:
```ruby
sayi=3
puts "alternatif if kullanımı" if sayi<10
#puts "unless alternatif" unless sayi>100
```

```
alternatif if kullanımı
```

case yapısı

- when ile verilen seçeneklerden hangisi doğru ise onu gerçekleştirir.
- when içinde aralık dışında düzenli ifadeler(regular expression) de verilebilir.

In [58]:
```ruby
#print "Ne kadar benzin kaldı(%)? "
#benzin = gets.chomp.to_i
benzin=0
durum=case benzin
    when 0
    "benzin miktarı: #{benzin} -> yolda kaldınız"
    when (1..15)
    "benzin miktarı: #{benzin} -> benzin istasyonuna gidin"
    when (16..100)
    "benzin miktarı: #{benzin} -> yola devam edebilirsiniz"
    else
    "hatalı deger girdisi !!"
    end

puts durum
```

```
benzin miktarı: 0 -> yolda kaldınız
```

times Metodu

- Integer sınıfından nesneler ile kullanılan ve verilen sayı kadar bir işin yapılmasını sağlayan döngü türüdür.

In [59]:
```ruby
10.times do
    puts "I will not talk in class!"
end
```

```
I will not talk in class!
I will not talk in class!
I will not talk in class!
I will not talk in class!
I will not talk in class!
I will not talk in class!
I will not talk in class!
I will not talk in class!
I will not talk in class!
I will not talk in class!
10
```

Out[59]:

In [60]:
```
10.times do |i| # döngü değişkeni olarak i
    puts i*2
end
```

```
0
2
4
6
8
10
12
14
16
18
```

Out[60]: 10

In [61]:
```
20.times{ |i| puts 2**i } # 2'nin kuvvetlerini yazdır
```

```
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
```

Out[61]: 20

for ve each Döngüleri

- for ve each döngüleri bir veri seti veya aralık üzerinde dolaşmayı sağlar.
- Yazım biçimi farklı olsa da aynı işleve sahiplerdir

In [65]:
```
for i in 1..10
  puts "#{i}"
end
```

```
1
2
3
4
5
6
7
8
9
10
```

Out[65]: 1..10

In [64]:
```ruby
(1..10).each do |i|
    puts i
end
```

```
1
2
3
4
5
6
7
8
9
10
```

Out[64]: 1..10

In [66]:
```ruby
dizi = [2,3,5,7,11]
dizi.each { |i| puts i }
```

```
2
3
5
7
11
```

Out[66]: [2, 3, 5, 7, 11]

In [67]:
```ruby
# döngü icinde BREAK kosulu saglandıgında döngü devre disi kalir.
i = 0
while i < 5 do
    break if i == 3
    puts "i = #{i}"
    i += 1
end
```

```
i = 0
i = 1
i = 2
```

In [ ]: