Python Programming

IF Karar Yapıları

Control Structures

if *condition*:
    statements
[elif *condition*:
    statements] ...
else:
    statements

while *condition*:
    statements

for *var* in *sequence*:
    statements

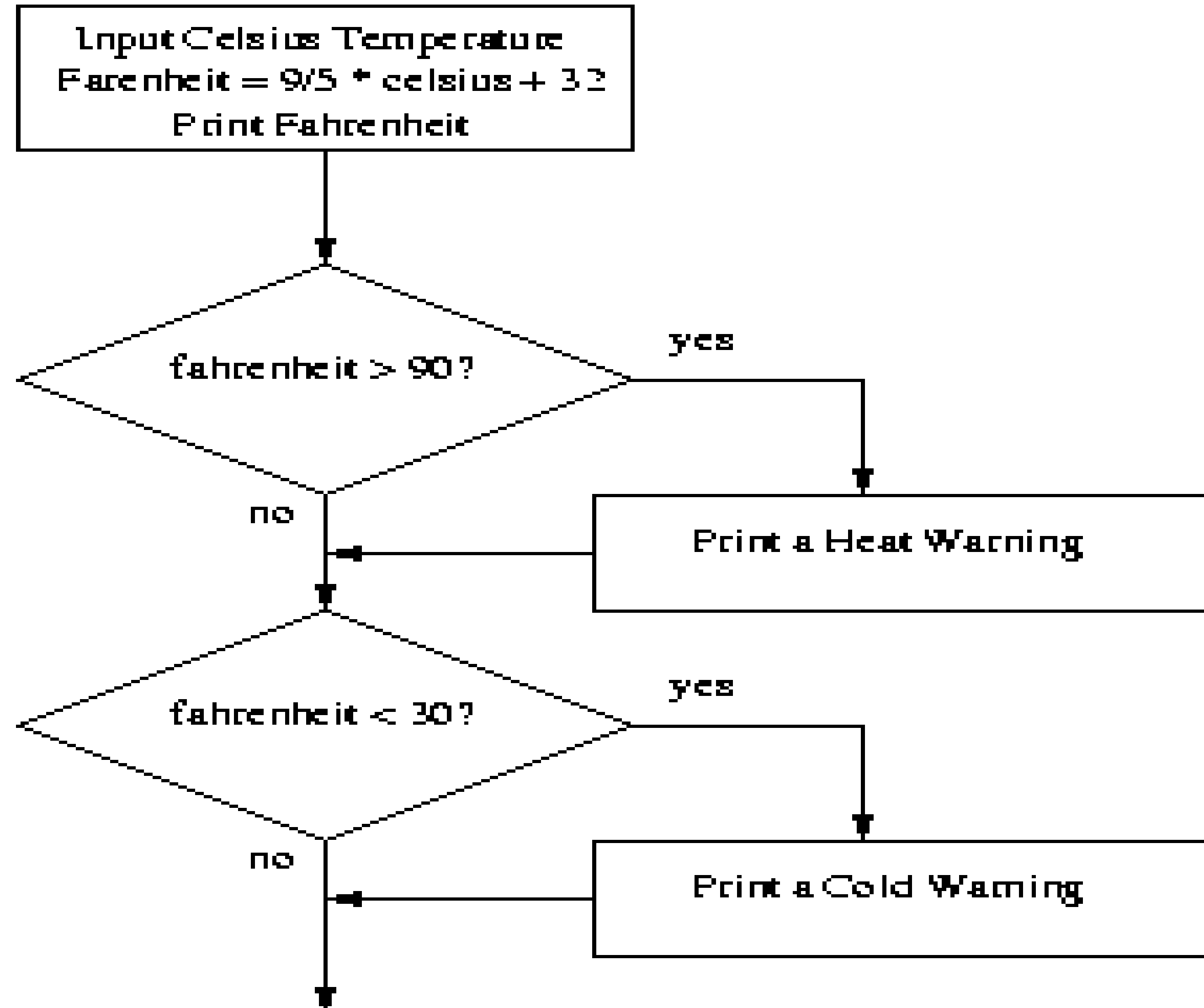
break
continue

Simple Decisions

- *Kontrol yapıları, sıralı program akışını değiştirmemize izin verir.*
- Bu bölümde, bir programın farklı durumlar için farklı komut dizilerini yürütmesine izin veren veya programın uygun bir eylem biçimini "seçmesine" izin veren ifadeler olan karar yapılarını öğreneceğiz.

Example: Temperature Warnings

- Bu yeni algoritmanın sonunda iki karar var.
- Girinti, bir adımın yalnızca önceki satırda listelenen koşul doğruysa gerçekleştirilmesi gerektiğini belirtir.

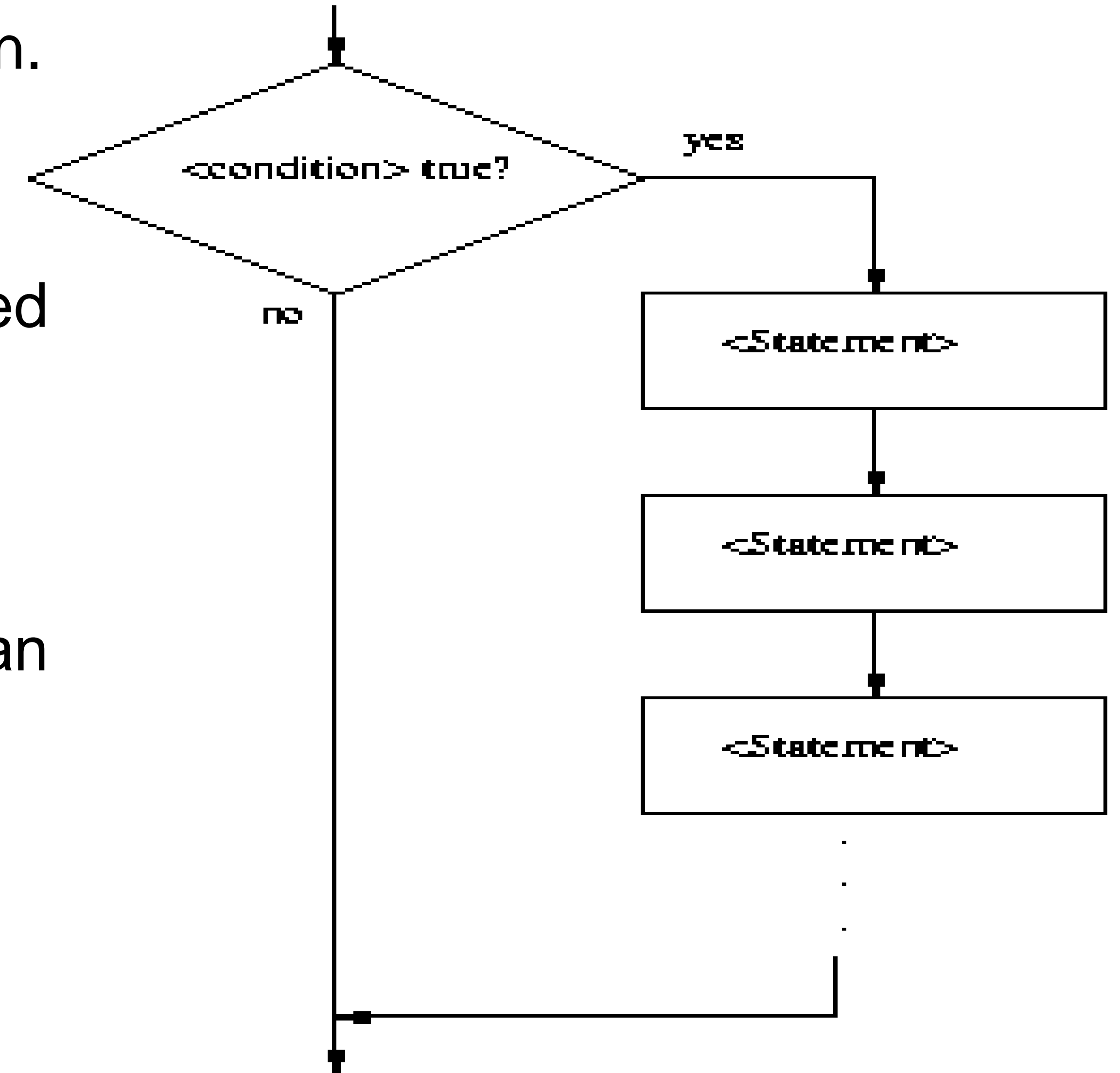


Example: Temperature Warnings

```
# convert2.py
#      A program to convert Celsius temps to Fahrenheit.
#      This version issues heat and cold warnings.

def main():
    celsius = eval(input("What is the Celsius temperature? "))
    fahrenheit = 9 / 5 * celsius + 32
    print("The temperature is", fahrenheit, "degrees fahrenheit.")
    if fahrenheit >= 32 and fahrenheit <= 90:
        print(" Evryting is Okey")
    elif fahrenheit > 90:
        print("It's really hot out there, be careful!")
    elif fahrenheit < 32:
        print("Brrrrr. Be sure to dress warmly")

main()
```

Example: Temperature Warnings

- The Python `if` statement is used to implement the decision.
- `if <condition>:`
  `   <body>`
- The body is a sequence of one or more statements indented under the `if` heading.
- if'in anlamı açık olmalıdır.
- Öncelikle başlıktaki durum değerlendirilir.
- Koşul doğruysa, gövdedeki ifade dizisi yürütülür ve ardından kontrol, programdaki bir sonraki ifadeye geçer.
- Koşul yanlışsa, gövdedeki ifadeler atlanır ve programdaki kontrol bir sonraki ifadeye geçer.
- If gövdesi, koşula bağlı olarak yürütülür veya yürütülmez. Her durumda, kontrol if'den sonraki ifadeye geçer.Bu tek yönlü veya basit bir karardır.



Forming Simple Conditions

- What does a condition look like?
- At this point, let's use simple comparisons.
- `<expr> <relop> <expr>`
- `<relop>` is short for *relational operator*

Python	Mathematics	Meaning
<	<	Less than
<=	≤	Less than or equal to
==	=	Equal to
>=	≥	Greater than or equal to
>	>	Greater than
!=	≠	Not equal to

Forming Simple Conditions

- Eşitlik için == kullanımına dikkat edin.
- Python, atamayı belirtmek için = kullandığından, eşitlik kavramı için farklı bir sembol gerekir.Yaygın bir hata, koşullarda = kullanmaktır!
- Koşullar, sayıları veya dizeleri karşılaştırabilir.
- Dizeleri karşılaştırırken sıralama sözlükseldir, yani dizeler temeldeki Unicode'a göre sıralanır. Bu nedenle, tüm büyük harfler küçük harflerden önce gelir. (“Bbb”, “aaaa”dan önce gelir)
- Koşullar, adını İngiliz matematikçi George Boole'dan alan Boole ifadelerine dayanmaktadır.
- Bir Boole ifadesi değerlendirildiğinde, koşulun geçerli olduğu durumda bir true değeri üretir veya bir koşulun geçerli olmadığı durumlarda ise false üretir
- Bazı bilgisayar dilleri "doğru" ve "yanlış"ı temsil etmek için 1 ve 0 kullanır.

Example: Conditional Program Execution

- There are several ways of running Python programs.
  - Bazı modüller doğrudan çalıştırılmak üzere tasarlanmıştır. Bunlara program veya betik adı verilir.
  - Diğerleri içe aktarılmak ve başka programlar tarafından kullanılmak üzere yapılır. Bunlara kütüphane adı verilir.
  - Bazen hem bağımsız bir program hem de bir kütüphane olarak kullanılabilen bir melez programlama oluşturmak istenbilir.

Example: Conditional Program Execution

- Bir program yüklendikten sonra başlatmak istediğimizde, kodun altına main() satırını ekleriz.
- Python, içe aktarma işlemi sırasında programın satırlarını değerlendirdiğinden, mevcut programlarımız etkileşimli bir Python oturumuna veya başka bir Python programına içe aktarıldığında da çalışır.
- Genel olarak, bir modülü içe aktardığımızda çalışmasını istemiyoruz!
- Tek başına çalıştırılabilen veya bir kitaplık olarak yüklenebilen bir programda, alttaki main çağrısı koşullu yapılmalıdır, örn.
-

```
if <condition>:
    main()
```

Example: Conditional Program Execution

- Bir modül içe aktarıldığında Python, içe aktarılan modülün adı olması için modülde modul.name adlı özel bir değişken oluşturur.
-

```
import math
math.name
```

Math kütüphanesi içeriği:

- math.e Returns Euler's number (2.7182...)
- math.pi Returns PI (3.1415...)
- math.sqrt(number) Returns the square root of the number
- math.tan(number) Returns the tangent of the number
- math.sin() Returns the sine of a number
- math.log() Returns the natural logarithm of a number, or the logarithm of number to base
- math.log10() Returns the base-10 logarithm of x
- math.log1p() Returns the natural logarithm of 1+x
- math.log2() Returns the base-2 logarithm of x

Two-Way Decisions

```
import math
def main():
    print("This program finds the real solutions to a quadratic")

    a, b, c = eval(input("\nPlease enter the coefficients (a, b, c): "))
    d=b * b - 4 * a * c
    if d>=0:
        discRoot = math.sqrt(d)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2)
    if d<0:
        print("Result is negative")

    d1=math.sin(math.pi/2)
    print("\nThe solutions are:", d1)
main()
```

Two-Way Decisions

- We can check for this situation. Here's our first attempt.

```
# quadratic2.py
#    A program that computes the real roots of a quadratic equation.
#    Bad version using a simple if to avoid program crash

import math

def main():
    print("This program finds the real solutions to a quadratic\n")

    a, b, c = eval(input("Please enter the coefficients (a, b, c): "))

    discrim = b * b - 4 * a * c
    if discrim >= 0:
        discRoot = math.sqrt(discrim)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2)
```

Two-Way Decisions

- We first calculate the discriminant (b^2-$4ac$) and then check to make sure it's nonnegative. If it is, the program proceeds and we calculate the roots.
- Look carefully at the program. What's wrong with it? Hint: What happens when there are no real roots?
-
```
This program finds the real solutions to a quadratic

Please enter the coefficients (a, b, c): 1,1,1
>>>
```
- This is almost worse than the version that crashes, because we don't know what went wrong!

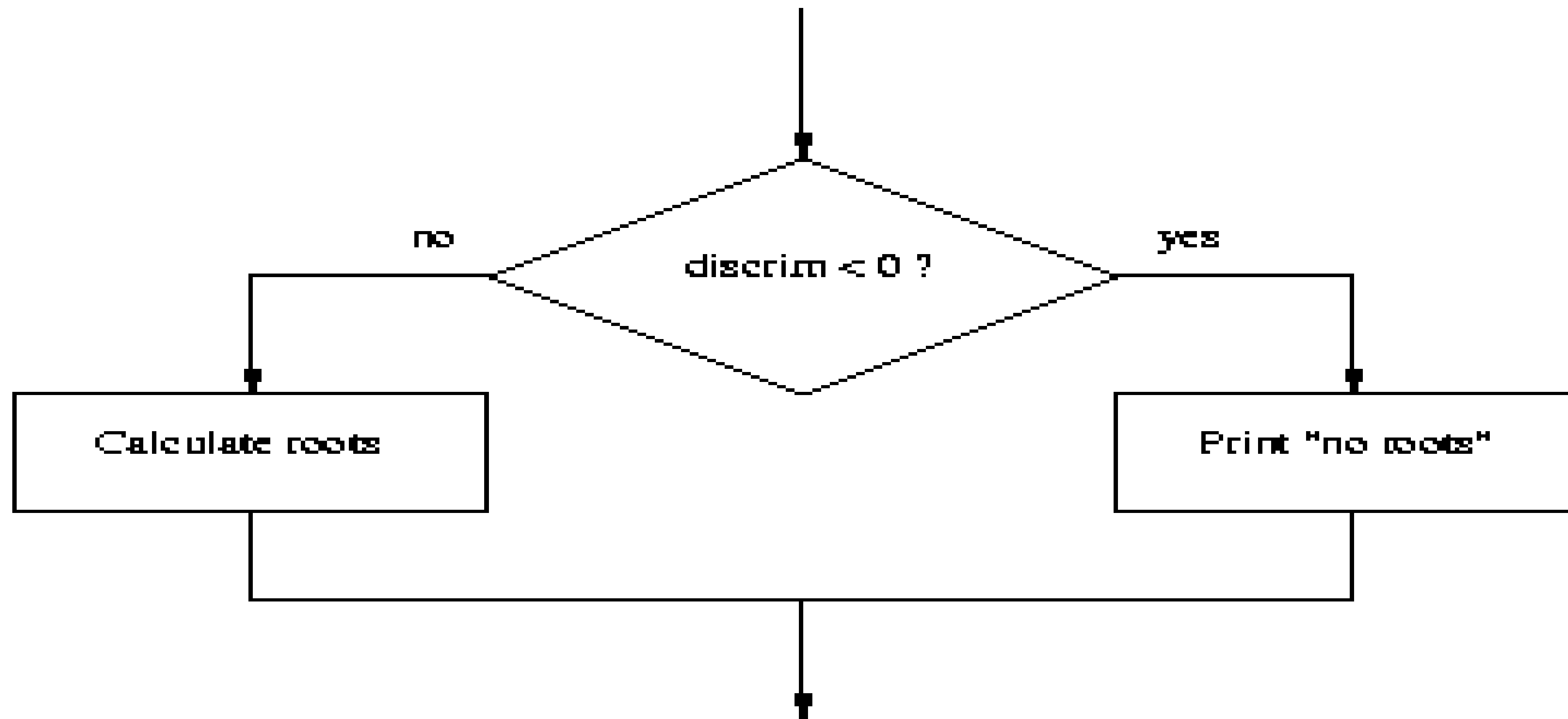
Two-Way Decisions

- We could add another `if` to the end:

```
if discrim < 0:
    print("The equation has no real roots!" )
```

- This works, but feels wrong. We have two decisions, with *mutually exclusive* outcomes (if `discrim >= 0` then `discrim < 0` must be false, and vice versa).

Two-Way Decisions



Two-Way Decisions

- In Python, a two-way decision can be implemented by attaching an `else` clause onto an `if` clause.
- This is called an `if-else` statement:

```
if <condition>:
    <statements>
else:
    <statements>
```

Two-Way Decisions

- When Python first encounters this structure, it first evaluates the condition. If the condition is true, the statements under the `if` are executed.
- If the condition is false, the statements under the `else` are executed.
- In either case, the statements following the `if-else` are executed after either set of statements are executed.

Two-Way Decisions

```
# quadratic3.py
#    A program that computes the real roots of a quadratic equation.
#    Illustrates use of a two-way decision

import math
def main():
    print "This program finds the real solutions to a quadratic\n"

    a, b, c = eval(input("Please enter the coefficients (a, b, c): "))

    discrim = b * b - 4 * a * c
    if discrim < 0:
        print("\nThe equation has no real roots!")
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print ("\nThe solutions are:", root1, root2 )

main()
```

Two-Way Decisions

```
>>>
This program finds the real solutions to a quadratic
Please enter the coefficients (a, b, c): 1,1,2
The equation has no real roots!

>>>
This program finds the real solutions to a quadratic
Please enter the coefficients (a, b, c): 2, 5, 2
The solutions are: -0.5 -2.0
```

Multi-Way Decisions

- The newest program is great, but it still has some quirks!

```
This program finds the real solutions to a
quadratic

Please enter the coefficients (a, b, c): 1,2,1
The solutions are: -1.0 -1.0
```

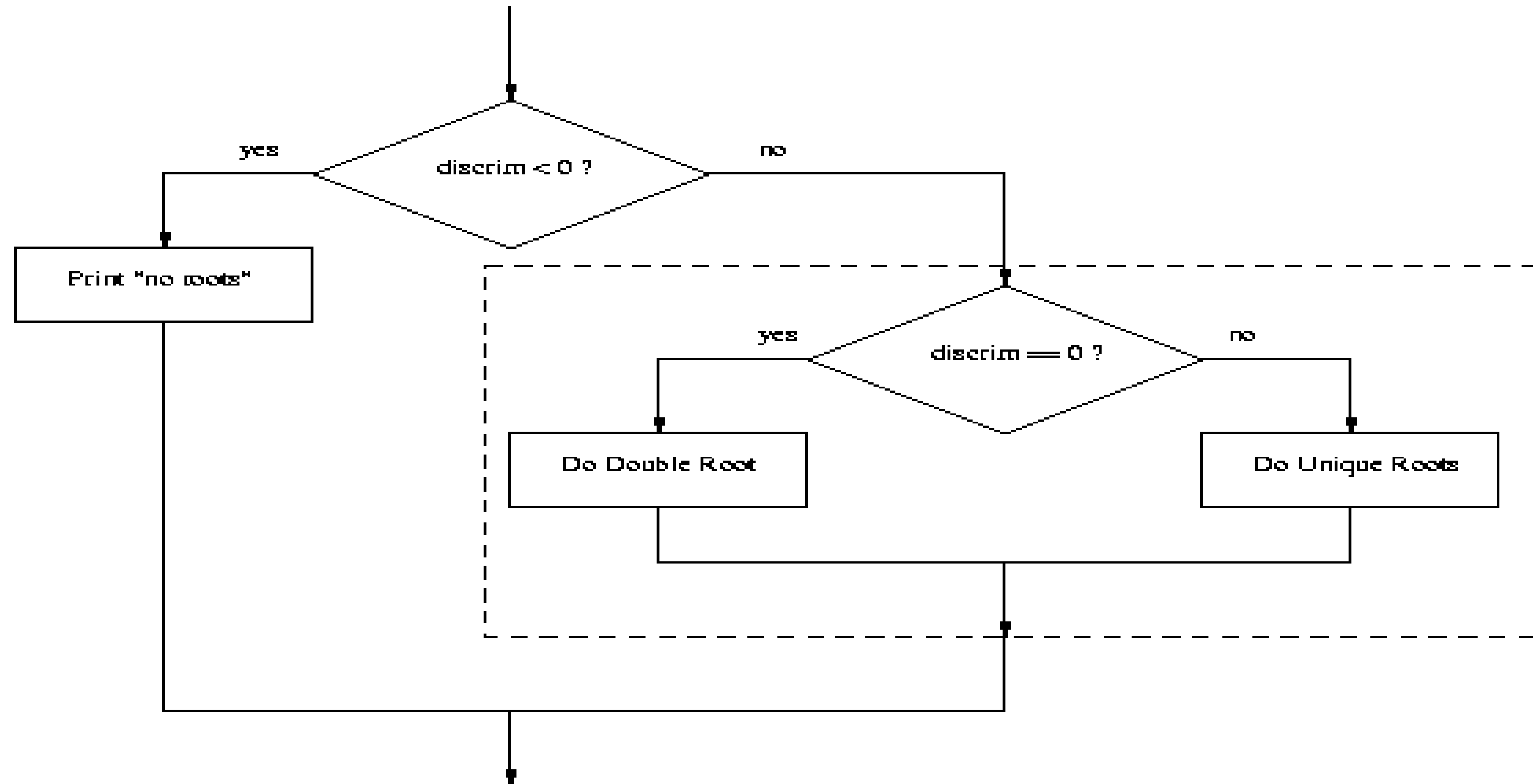
Multi-Way Decisions

- While correct, this method might be confusing for some people. It looks like it has mistakenly printed the same number twice!
- Double roots occur when the discriminant is exactly 0, and then the roots are $-b/2a$.
- It looks like we need a three-way decision!
-
```
Check the value of discrim
   when < 0: handle the case of no roots
   when = 0: handle the case of a double root
   when > 0: handle the case of two distinct roots
```
- We can do this with two if-else statements, one inside the other.
- Putting one compound statement inside of another is called *nesting*.

Multi-Way Decisions

```
if discrim < 0:
    print("Equation has no real roots")
else:
    if discrim == 0:
        root = -b / (2 * a)
        print("There is a double root at", root)
    else:
        # Do stuff for two roots
```

Multi-Way Decisions



Multi-Way Decisions

- Imagine if we needed to make a five-way decision using nesting. The `if-else` statements would be nested four levels deep!
- There is a construct in Python that achieves this, combining an `else` followed immediately by an `if` into a single `elif`.

-
```
if <condition1>:
   <case1 statements>
elif <condition2>:
   <case2 statements>
elif <condition3>:
   <case3 statements>
…
else:
   <default statements>
```

Multi-Way Decisions

- This form sets of any number of mutually exclusive code blocks.
- Python evaluates each condition in turn looking for the first one that is true. If a true condition is found, the statements indented under that condition are executed, and control passes to the next statement after the entire `if-elif-else`.
- If none are true, the statements under `else` are performed.
- The `else` is optional. If there is no `else`, it’s possible no indented block would be executed.

Multi-Way Decisions

```
# quadratic4.py
#    Illustrates use of a multi-way decision

import math

def main():
    print("This program finds the real solutions to a quadratic\n")
    a, b, c = eval(input("Please enter the coefficients (a, b, c): "))

    discrim = b * b - 4 * a * c
    if discrim < 0:
        print("\nThe equation has no real roots!")
    elif discrim == 0:
        root = -b / (2 * a)
        print("\nThere is a double root at", root)
    else:
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2 )
```

Exception Handling

- İkinci dereceden programda, negatif bir sayının karekökünü almaktan kaçınmak için karar yapılarını kullandık, böylece bir hatadan kaçındık.
- Bu, pek çok program için geçerlidir: karar yapıları, nadir fakat olası hatalara karşı koruma sağlamak için kullanılır.
- İkinci dereceden örnekte, sqrt'yi çağırmadan önce verileri kontrol ettik.
- Bazen işlevler hataları kontrol eder ve işlemin başarısız olduğunu belirtmek için özel bir değer döndürür.
- Örneğin, farklı bir karekök işlemi, bir hatayı belirtmek için –1 döndürebilir (karekökler hiçbir zaman negatif olmadığından, bu değerin benzersiz olacağını biliyoruz).

Exception Handling

- ```
  discRt = otherSqrt(b*b - 4*a*c)
  if discRt < 0:
      print("No real roots.“)
  else:
      ...
  ```
- Bazen programlar, özel durumlar için o kadar çok kontrol alır ki, algoritmayı takip etmek zorlaşır.
- Programlama dili tasarımcıları, bu tasarım problemini çözmek için istisna işlemeyi ele alan bir mekanizma geliştirdiler.
  ```

Exception Handling

- Programcı, program çalışırken ortaya çıkan hataları yakalayan ve bunlarla ilgilenen kod yazabilir, yani "Bu adımları uygular ve herhangi bir sorun ortaya çıkarsa, bir şekilde halledin.
- "Bu yaklaşım, algoritmadaki her adımda açık kontrol yapma ihtiyacını ortadan kaldırır.

Exception Handling

- The `try` statement has the following form:

```
try:
   <body>
except <ErrorType>:
   <handler>
```

- Python bir try ifadesi ile karşılaştığında, gövde içindeki ifadeleri yürütmeye çalışır.
- Hata yoksa kontrol, try…except'ten sonra bir sonraki ifadeye geçer.

Exception Handling

```
# quadratic5.py
#    A program that computes the real roots of a quadratic equation.
#    Illustrates exception handling to avoid crash on bad inputs

import math
def main():
    print("This program finds the real solutions to a quadratic\n")

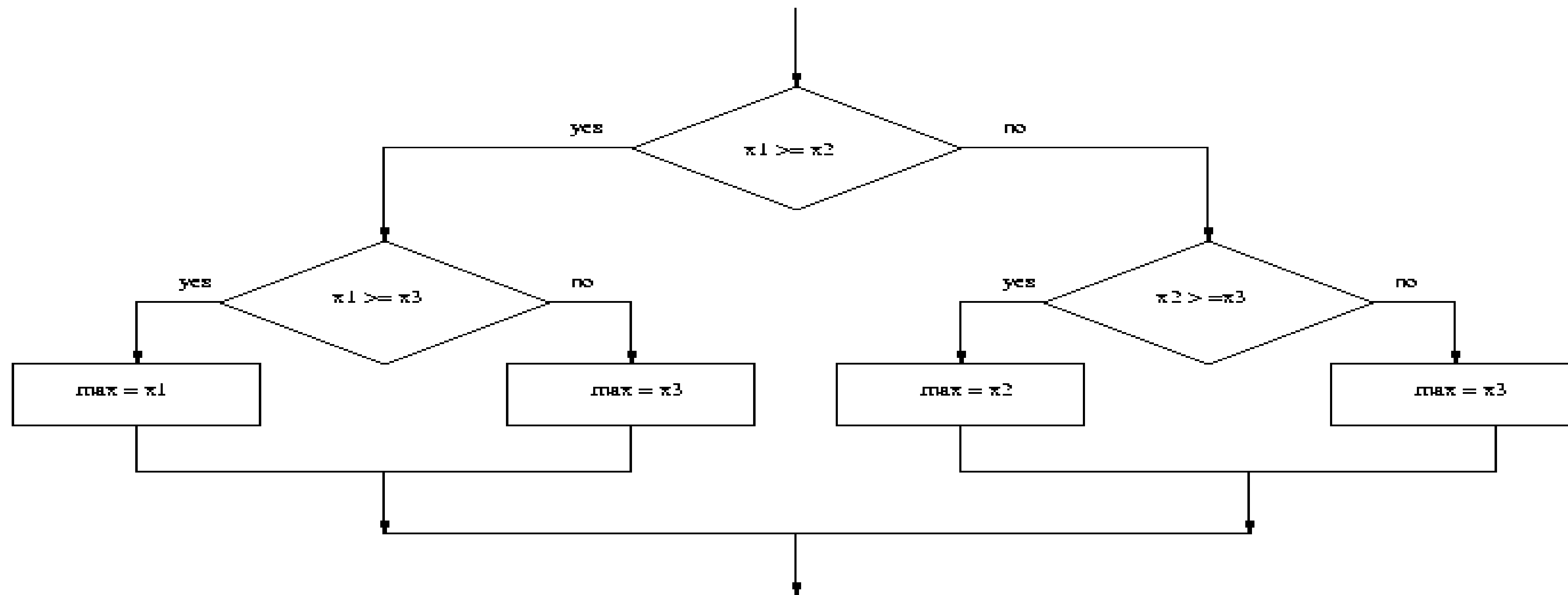
    try:
        a, b, c = eval(input("Please enter the coefficients (a, b, c): "))
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2)
    except ValueError:
        print("\nNo real roots")
main()
```

Exception Handling

- If an error occurs while executing the body, Python looks for an except clause with a matching error type. If one is found, the handler code is executed.
- The original program generated this error with a negative discriminant:

```
Traceback (most recent call last):
  File "C:\Documents and Settings\Terry\My Documents\Teaching\W04\CS120\Textbook\code\chapter3\quadratic.py", line 21, in -toplevel-
    main()
  File "C:\Documents and Settings\Terry\My Documents\Teaching\W04\CS 120\Textbook\code\chapter3\quadratic.py", line 14, in main
    discRoot = math.sqrt(b * b - 4 * a * c)
ValueError: math domain error
```

Exception Handling

- The last line, "`ValueError: math domain error`", indicates the specific type of error.
- Here's the new code in action:

```
This program finds the real solutions to a quadratic

Please enter the coefficients (a, b, c): 1, 1, 1

No real roots
```

- Instead of crashing, the exception handler prints a message indicating that there are no real roots.

Exception Handling

- The `try…except` can be used to catch *any* kind of error and provide for a graceful exit.
- In the case of the quadratic program, other possible errors include not entering the right number of parameters ("`unpack tuple of wrong size`"), entering an identifier instead of a number (`NameError`), entering an invalid Python expression (`TypeError`).
- A single `try` statement can have multiple `except` clauses.

Exception Handling

```
# quadratic6.py
import math

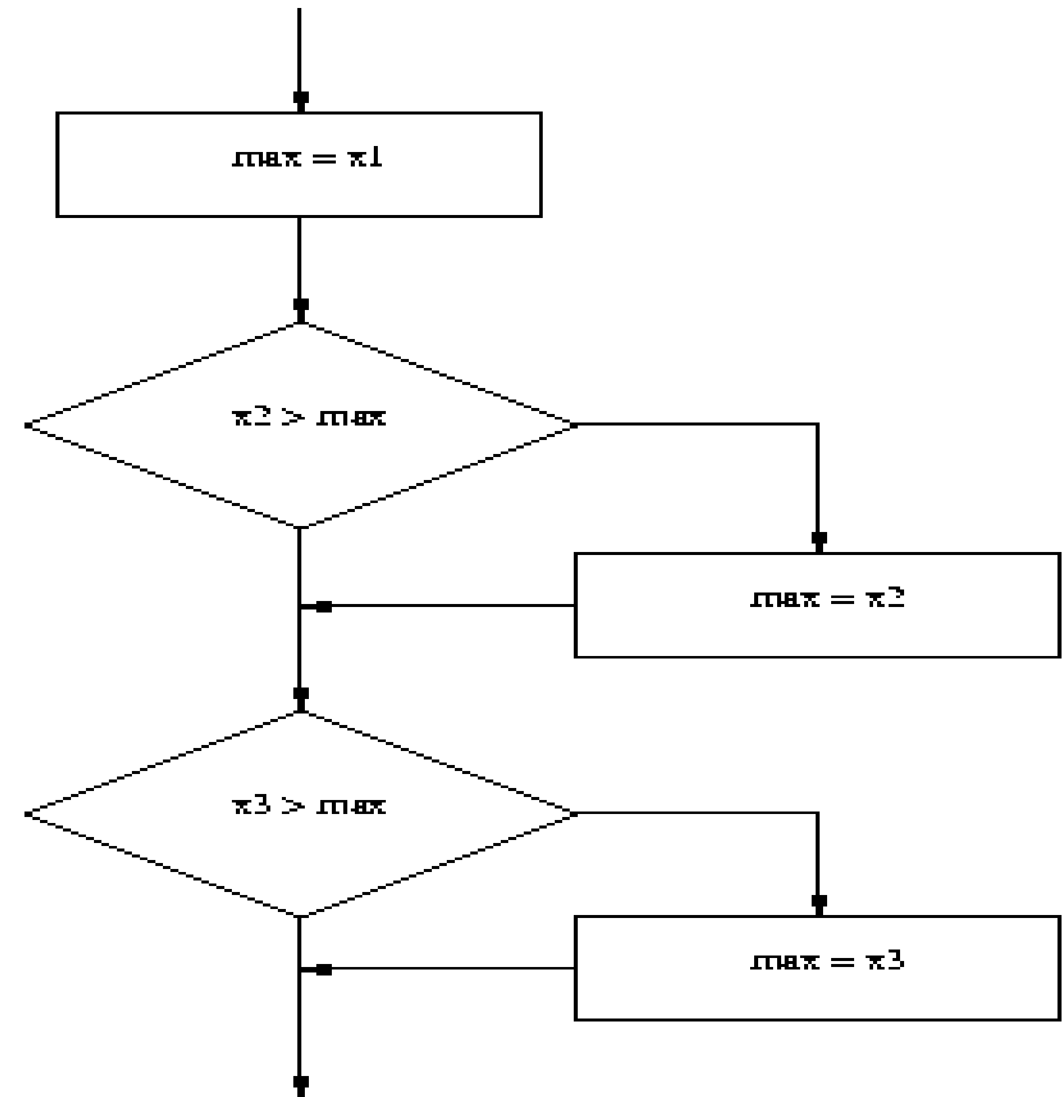
def main():
    print("This program finds the real solutions to a quadratic\n")

    try:
        a, b, c = eval(input("Please enter the coefficients (a, b, c): "))
        discRoot = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2 )
    except ValueError as excObj:
        if str(excObj) == "math domain error":
            print("No Real Roots")
        else:
            print("You didn't give me the right number of coefficients.")
    except NameError:
        print("\nYou didn't enter three numbers.")
    except TypeError:
        print("\nYour inputs were not all numbers.")
    except SyntaxError:
        print("\nYour input was not in the correct form. Missing comma?")
    except:
        print("\nSomething went wrong, sorry!")

main()
```

Exception Handling

- The multiple `excepts` act like `elifs`. If an error occurs, Python will try each `except` looking for one that matches the type of error.
- The bare `except` at the bottom acts like an `else` and catches any errors without a specific match.
- If there was no bare `except` at the end and none of the `except` clauses match, the program would still crash and report an error.
- Exceptions themselves are a type of object.
- If you follow the error type with an identifier in an `except` clause, Python will assign that identifier the actual exception object.

Study in Design: Max of Three

- Artık karar yapılarına sahip olduğumuza göre, daha karmaşık programlama problemlerini çözebiliriz. Olumsuz tarafı, bu programları yazmak zorlaşıyor!
- Üç sayıdan en büyüğünü bulmak için bir algoritmaya ihtiyacımız olduğunu varsayalım.

```
def main():
    x1, x2, x3 = eval(input("Please enter three values: "))
    # missing code sets max to the value of the largest
    print("The largest value is", max)
```

Strategy 1:
Compare Each to All

- This looks like a three-way decision, where we need to execute *one* of the following:

```
max = x1
max = x2
max = x3
```

- All we need to do now is preface each one of these with the right condition!

Strategy 1:Compare Each to All

- Let's look at the case where x1 is the largest.
-
```
if x1 >= x2 >= x3:
    max = x1
```
- Is this syntactically correct?
  - Many languages would not allow this *compound condition*
  - Python does allow it, though. It's equivalent to
    $x1 \geq x2 \geq x3$.
- Whenever you write a decision, there are two crucial questions:
  - When the condition is true, is executing the body of the decision the right action to take?
    - x1 is at least as large as x2 and x3, so assigning max to x1 is OK.
    - Always pay attention to borderline values!!

Strategy 1: Compare Each to All

- Secondly, ask the converse of the first question, namely, are we certain that this condition is true in all cases where x1 is the max?
  - Suppose the values are 5, 2, and 4.
  - Clearly, x1 is the largest, but does x1 ≥ x2 ≥ x3 hold?
  - We don’t really care about the relative ordering of x2 and x3, so we can make two separate tests: x1 >= x2 *and* x1 >= x3.

Strategy 1: Compare Each to All

- We can separate these conditions with *and*!

```
if x1 >= x2 and x1 >= x3:
    max = x1
elif x2 >= x1 and x2 >= x3:
    max = x2
else:
    max = x3
```

- We're comparing each possible value against all the others to determine which one is largest.

Strategy 1:
Compare Each to All

- En fazla beş değer bulmaya çalışsaydık ne olurdu?
- Her biri birlikte dört koşuldan oluşan dört Boole ifadesine ihtiyacımız olacak.

Strategy 2: Decision Tree

- We can avoid the redundant tests of the previous algorithm using a *decision tree* approach.
- Suppose we start with `x1 >= x2`. This knocks either `x1` or `x2` out of contention to be the max.
- If the conidition is true, we need to see which is larger, `x1` or `x3`.



Strategy 2: Decision Tree

- ```
  if x1 >= x2:
     if x1 >= x3:
        max = x1
     else:
        max = x3
  else:
     if x2 >= x3:
        max = x2
     else
        max = x3
  ```
- This approach makes exactly two comparisons, regardless of the ordering of the original three variables.
- However, this approach is more complicated than the first. To find the max of four values you’d need `if-else`s nested three levels deep with eight assignment statements.
  ```

Strategy 3: Sequential Processing

- How would you solve the problem?
- You could probably look at three numbers and just *know* which is the largest. But what if you were given a list of a hundred numbers?
- One strategy is to scan through the list looking for a big number. When one is found, mark it, and continue looking. If you find a larger value, mark it, erase the previous mark, and continue looking.



Strategy 3: Sequential Programming

- This idea can easily be translated into Python.

```
max = x1
if x2 > max:
    max = x2
if x3 > max:
    max = x3
```

- This process is repetitive and lends itself to using a loop.
- We prompt the user for a number, we compare it to our current max, if it is larger, we update the max value, repeat.

Strategy 3: Sequential Programming

```
# maxn.py
#    Finds the maximum of a series of numbers

def main():
    n = eval(input("How many numbers are there? "))

    # Set max to be the first value
    max = eval(input("Enter a number >> "))

    # Now compare the n-1 successive values
    for i in range(n-1):
        x = eval(input("Enter a number >> "))
        if x > max:
            max = x

    print("The largest value is", max)
```

Strategy 4:Use Python

- Python has a built-in function called `max` that returns the largest of its parameters.

```
def main():
    x1, x2, x3 = eval(input("Please enter three values: "))
    ad=max(x1, x2, x3)
    print("The largest value is",ad )
main()
```

Some Lessons

- **Genellikle bir sorunu çözmenin birden fazla yolu vardır.**
  - Aklınıza gelen ilk fikri kodlamak için acele etmeyin.
  - Tasarım hakkında düşünün ve soruna yaklaşmanın daha iyi bir yolu olup olmadığını sorun.
  - İlk göreviniz doğru bir algoritma bulmaktır. Ardından netlik, basitlik, verimlilik, ölçeklenebilirlik ve zarafet için çaba gösterin.
- **Bilgisayar gibi düşünün.**
  - Bir algoritma formüle etmenin en iyi yollarından biri, kendinize sorunu nasıl çözeceğinizi sormaktır.
  - Bu doğrudan yaklaşım genellikle basit, açık ve yeterince etkilidir.

Some Lessons

- **Genellik iyidir.**
  - Daha genel bir problemin ele alınması, özel bir durum için daha iyi bir çözüme yol açabilir.
  - Eğer max of n programını yazmak, max of three kadar kolaysa, daha genel programı yazın çünkü diğer durumlarda yararlı olma olasılığı daha yüksektir.
- **Tekerleği yeniden icat etmeyin.**
  - Çözmeye çalıştığınız sorun, birçok kişinin karşılaştığı bir sorunsa, bunun için zaten bir çözüm olup olmadığını öğrenin!
  - Programlamayı öğrenirken, programları sıfırdan tasarlamak harika bir deneyimdir!
  - Gerçekten uzman programcılar ne zaman ödünç alacaklarını bilirler.

Usage Notes

- A lot of slides are adopted from the presentations and documents published on internet by experts who know the subject very well.
- I would like to thank who prepared slides and documents.
- Also, these slides are made publicly available on the web for anyone to use
- If you choose to use them, I ask that you alert me of any mistakes which were made and allow me the option of incorporating such changes (with an acknowledgment) in my set of slides.

Sincerely,

Dr. Cahit Karakuş

cahitkarakus@esenyurt.edu.tr