MY470 Computer Programming

Control Flow in Python

Week 3 Lecture, MT 2017

Overview

- What is control flow?
- Conditional statements
- Iteration
- List comprehensions
- Examples
  - Exhaustive enumeration
  - Bisection search
  - Newton-Raphson algorithm

From Last Week: Straight-Line Programs

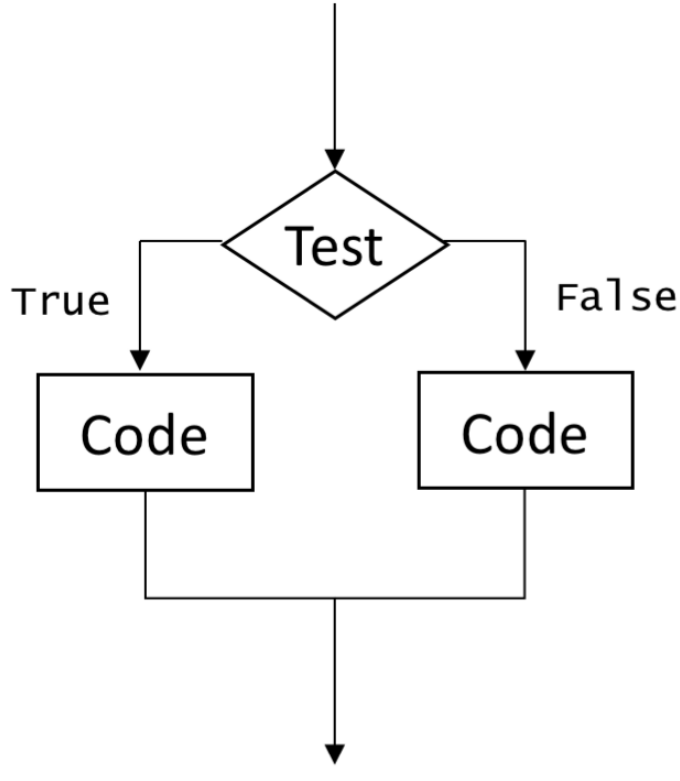
In [45]:
```
s = 'All animals are equal, but some animals are more equal than others.'
s = s.rstrip('.').lower()
s_tokens = s.split()
print('There are', len(s_tokens), 'words in the sentence.')
```

```
There are 12 words in the sentence.
```

Control Flow

- Control flow is the order in which statements are executed or evaluated
- In Python, there are three main categories of control flow:
  - **Branches** (conditional statements) – execute only if some condition is met
  - **Loops** (iteration) – execute repeatedly
  - Function calls – execute a set of distant statements and return back to the control flow

Conditional Statements



Conditional Statements: `if`

```
if *Boolean expression*:
    *block of code*
```

In [46]:
```
x = input('How old are you? ')
if int(x) >= 25:
    print("Ah, I see, you are a mature student.")
```

```
How old are you? 26
Ah, I see, you are a mature student.
```

Indentation in Python Code

- Indentation is semantically meaningful in Python
- You can use tabs or spaces (https://www.youtube.com/watch?v=SsoOG6ZeyUI)

- Obviously(!), tabs are preferable
- However, it does not really matter in Jupyter as Jupyter converts tabs to spaces by default

Conditional Statements: `if-else`

```
if *Boolean expression*:
    *block of code*
else:
    *block of code*
```

In [47]:
```
x = 5
if x%2==0:
    print("Even")
else:
    print("Odd")
print('Problem solved!')
```

```
Odd
Problem solved!
```

Conditional Statements: `if-elif-else`

```
if *Boolean expression*:
    *block of code*
elif *Boolean expression*:
    *block of code*
else:
    *block of code*
```

In [48]:
```
x = -2
if x > 0:
    print('Positive')
elif x < 0:
    print('Negative')
else:
    print('Zero')
```

```
Negative
```

Conditional Statements Are Evaluated Sequentially

Hence, it makes sense to start with the most likely one. This could make your code faster!

In [49]:
```
correct = 25

guess = int(input("Guess which number from 1 to 100 I'm thinking of? "))

if guess > correct + 10 or guess < correct - 10:
    print("You are quite far. Try again.")
elif guess != correct:
    print("You are very close. Try again.")
else:
    print("That's right!")
```

```
Guess which number from 1 to 100 I'm thinking of? 30
You are very close. Try again.
```

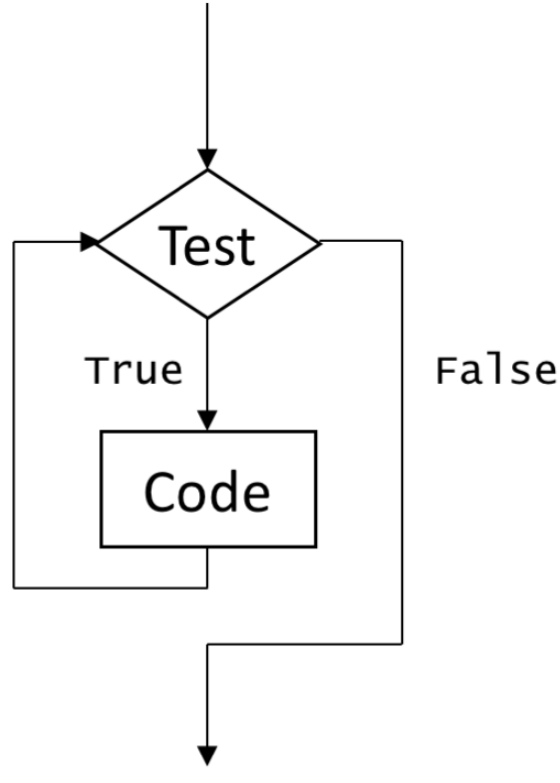
Nested Conditional Statements

Nesting conditional statements is often a question of style. As always, clarity and speed should be your major considerations!

In [50]:
```
x = -100

if type(x)==int or type(x)==float:
    if x > 0:
        print('This is a non-negative number')
    else:
        print('This is a negative number')
elif type(x)==str:
    print('This is a string.')
else:
    print("I don't now what this is.")
```

```
This is a negative number
```

Iteration



Iteration: `while` vs. `for`

```
while *Boolean expression*:
    *block of code*
```

```
for *element* in *sequence*:
    *block of code*
```

Iteration: `while` with decrementing function

The decrementing function is a function that maps variables to an integer that is initially non-negative but that decreases with every pass through the loop; the loop ends when the integer is 0.

In [51]:
```
# decrementing function: 5 - x
x = 0
while x < 5:
    x += 1
    print(x)
```

```
1
2
3
4
5
```

Iteration: `while` with conditional statements

In [52]:
```
correct = 25
repeat = True

while repeat:
    guess = int(input("Guess which number from 1 to 100 I'm thinking of? "))

    if guess > correct + 10 or guess < correct - 10:
        print("You are quite far. Try again.")
    elif guess != correct:
        print("You are very close. Try again.")
    else:
        print("That's right!")
        repeat=False
```

```
Guess which number from 1 to 100 I'm thinking of? 70
You are quite far. Try again.
Guess which number from 1 to 100 I'm thinking of? 24
You are very close. Try again.
Guess which number from 1 to 100 I'm thinking of? 25
That's right!
```

Iteration: `for`

```
for *element* in *sequence*:
    *block of code*
```

In [53]:
```
for i in [1, 2, 3, 4, 5]:
    print(i, end=' ')
```

```
1 2 3 4 5
```

`range()`

- In-built function that produces an immutable ordered non-scalar object of type `range`
- Initiate as `range([start], stop, [step])`. If ommitted, `start = 0` and `step = 1`.
- Function produces progression of integers `[start, start + step, start + 2*step, ..., start + i*step]`
  - If step > 0, `start + i*step < stop`
  - If step < 0, `start + i*step > stop`

In [6]:
```
print(range(6))
print(list(range(6)))
```

```
range(0, 6)
[0, 1, 2, 3, 4, 5]
```

`range()` is essential for `for`-loops

In [7]:
```
for i in range(6):
    print(i, end=' ')
print()

for i in range(1,6):
    print(i, end=' ')
print()

for i in range(1,6,2):
    print(i, end=' ')
```

```
0 1 2 3 4 5
1 2 3 4 5
1 3 5
```

Indexing Lists with `range(len(L))`

In [55]:
```
mylist = ['a', 'b', 'c', 'd']
for i in range(len(mylist)):
     print('index', i, ':', mylist[i])
```

```
index 0 : a
index 1 : b
index 2 : c
index 3 : d
```

Iteration: `break` and `continue`

- Use `break` to exit a loop
- Use `continue` to go directly to next iteration

In [56]:
```
for i in range(5):
    if i==2:
        continue  # Now try with break
    print(i)
```

```
0
1
3
4
```

List Comprehensions

```
L = [*object, expression, or function* for *element* in *sequence*]
L = [*object, expression, or function* for *element* in *sequence* if *Boolean ex
pression*]
L = [*object, expression, or function* for *element* in *sequence* for *element2*
 in *sequence2*]
```

- Provide a concise way to create lists
- Faster because implemented in C
- Nested list comprehension can be somewhat confusing

List Comprehensions

In [57]:
```
print([x**2 for x in range(1,11)])

ans = []
for x in range(1,11):
    ans.append(x**2)
print(ans)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In [58]:
```
print([x**2 for x in range(1,11) if x%2==0])
print([x+y for x in ['a', 'b', 'c'] for y in ['1','2', '3']])
```

```
[4, 16, 36, 64, 100]
['a1', 'a2', 'a3', 'b1', 'b2', 'b3', 'c1', 'c2', 'c3']
```

Dictionary and Set Comprehensions

In [59]:
```
print({x: x**2 for x in range(1,11)})
print({x.lower(): y for x, y in [('A',1), ('b',2), ('C',2)]})

print({x.lower() for x in 'SomeRandomSTRING'})
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
{'a': 1, 'b': 2, 'c': 2}
{'s', 'o', 'n', 'd', 'm', 'e', 't', 'i', 'a', 'g', 'r'}
```

Example: Exhaustive Enumeration

In [60]:
```
# Find an approximation to the square root of a number

x = 2500

ans = 0

# Increment ans until all options exhausted
while ans**2 < abs(x):
    ans += 1

if ans**2 != abs(x):
    print(x, 'is not a perfect square')
else:
    print('Square root of', x, 'is', ans)
```

```
Square root of 2500 is 50
```

Exhaustive Enumeration

- Systematically enumerate all possible solutions until you get the right answer or run out of possibilities
- Example of **brute-force search** (aka **guess and check** strategy) — a general problem-solving technique in computer science
- Suprisingly useful as computers are quite fast these days!

Example: Approximation with Exhaustive Enumeration

In [61]:
```
# Find an approximation to the square root of a number using exhaustive enumeratio
n

x = 25

epsilon = 0.01  # Precision of approximation
step = epsilon**2

num_guess = 0 # Keep track of iteration steps
ans = 0

# Increment ans with step until close enough or until all options exhausted
while abs(ans**2 - x) >= epsilon and ans <= x:
    ans += step
    num_guess += 1

if abs(ans**2 - x) >= epsilon:
    print('Failed to find close approximation to the square root of', x)
else:
    print('Found', ans, 'to be close approximation to square root of', x)

print('num_guess =', num_guess)
```

```
Found 4.999000000001688 to be close approximation to square root of 25
num_guess = 49990
```

Bisection Search

- Start in the middle of the array, eliminate the half in which the answer cannot lie, and continue the search in the other half until you get the right answer or run out of possibilities
- Example of **divide and conquer** strategy — an algorithm-design paradigm in computer science
- Naturally implemented as a recursive procedure (covered next week)

Example: Approximation with Bisection Search

In [62]:
```
# Find an approximation to the square root of a number using bisection search

x = 25

epsilon = 0.01  # Precision of approximation
num_guess = 0 # Keep track of iteration steps

# Define interval for search
low = 0
high = max(1, x)

# Start in the middle
ans = (high + low) / 2

# Narrow down search interval until ans close enough
while abs(ans**2 - x) >= epsilon:
    if ans**2 < x:
        low = ans
    else:
        high = ans
    ans = (high + low) / 2
    num_guess += 1

print('Found', ans, 'to be close approximation to square root of', x)

print('num_guess =', num_guess)
```

```
Found 5.00030517578125 to be close approximation to square root of 25
num_guess = 13
```

Newton-Raphson Method for Finding Polynomial Roots

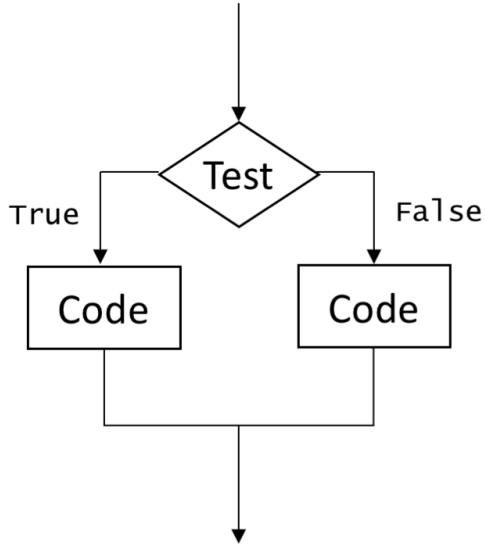
- $x^2 - 25$ is a polynomial p
- Newton proved a theorem that implies that if a is a an approximation to the root of $p = 0$, then $a - \frac{p(a)}{p'(a)}$ is a better approximation
- p' is the first derivative of p. For $p = x^2 - 25, p' = 2x$

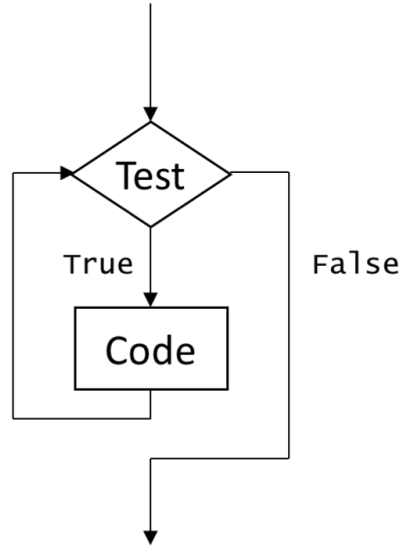
Example: Approximation with Newton-Raphson Method

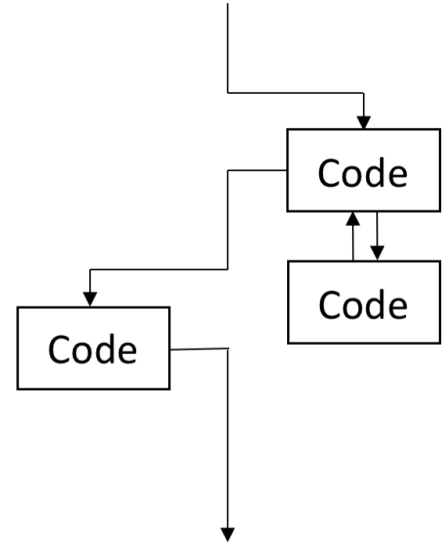
In [8]:
```
# Find an approximation to the square root of a number using Newton-Raphson method
# Find x such that x**2 - 25 is within epsilon of 0

k = 25

epsilon = 0.01  # Precision of approximation
num_guess = 0 # Keep track of iteration steps

# Initialize first guess
ans = k

# Use Newton's theorem until ans close enough
while abs(ans**2 - k) >= epsilon:
    ans = ans - ( (ans**2 - k) / (2*ans))
    num_guess += 1

print('Found', ans, 'to be close approximation to square root of', k)

print('num_guess =', num_guess)
```

```
Found 5.000023178253949 to be close approximation to square root of 25
num_guess = 5
```

Control Flow







- **Lab**: `for` loops and list comprehensions, including nested list comprehensions
- **Assignment**: Practice conditional statements and iteration on data
- **Next week**: Functions in Python