15-112
Fundamentals of Programming

Lecture 2:
Basic Building Blocks of Programming Continued
Basic Building Blocks

Statements
Tell the computer to do something. An instruction.

Data Types
Data is divided into different types.

Variables
Allows you to store data and access stored data.

Operators
Allows you to manipulate data.

Functions
Programs are structured using functions.

Conditional Statements
Executes statements if a condition is satisfied.

Loops
Execute a block of code multiple times.
One the menu today:

More on operators

More on functions

Conditional statements

Practice problem(s)
More on operators
More on operators

Arithmetic operators: +  −  *  /  //  **  %

Assignment operators: +=  -=  *=  /=  //=  %=

Comparison operators:  ==  !=  <  <=  >  >=
(takes two numerical values and produces bool value)

Boolean operators:  not  or  and
% Modulo operator

n % m means n mod m
% Modulo operator

\( n \% m \) means \( n \mod m \)

When \( n \) is positive: \( n \% m \) is the remainder when \( n \) is divided by \( m \)

When \( n \) is negative: add multiples of \( m \) to \( n \) until you are between 0 and \( m-1 \)
More on operators

%  Modulo operator

\[ n \% m \quad \text{means} \quad n \mod m \]

A couple of useful things you can do:

\[ n \% 1 \quad \text{the fractional part of } n \]

\[ n \% 2 \quad \text{parity of } n \]
More on operators

**Boolean operators:** not  or  and

- **not** boolean-expression
  - Flips the value of the expression.
  - `not ("123" == 123)`  `not (3 == 3.0)`

- **boolean-exp1 and boolean-exp2**
  - Evaluates to True only if both expressions are True.
  - `(("a" < "b") and ("b" < "z"))`

- **boolean-exp1 or boolean-exp2**
  - Evaluates to True only if at least one of the expressions is True.
  - `((False < True) or False)`
The rules correspond to how we use “and” and “or” in our daily lives.

I have an apple OR I have an orange.

I have an apple AND I have an orange.
More on operators

Operator Precedence

**Summary:** what you would expect!

or
and
not

comparison operators:  `==`, `!=`, `<`, `>`, ...

`+`, `-`

`*`, `/`, `//`, `%`

`**`

print(1 < 2 and 5 < 2 + 1 * 2)  
yuck!
More on functions
More on functions

A function in Python:

input(s) → \( f \) → output

In fact,

Python program = a function + other “helper” functions
Example problem:
Write a function that takes 2 integers as input and returns the maximum of the ones digit of the numbers.

```python
def max(x, y):
    # some code here

def onesDigit(x):
    # some code here

def largerOnesDigit(x, y):
    return max(onesDigit(x), onesDigit(y))
```
Write a function that takes an integer and returns its tens digit.

```python
def tensDigit(n):
    return (n // 10) % 10
```

**Hint:** If \( n \) is the input, think about the values \( n \mod 10 \) and \( n // 10 \)

Always test your function before you move on!
More on functions

Test function

def testTensDigit():
    assert(tensDigit(5) == 0)
    assert(tensDigit(95) == 9)
    assert(tensDigit(4321) == 2)
    assert(tensDigit(-1234) == 3)  # Fail
    print("Passed all tests!")

Make sure you select your test cases carefully!

Retry:

def tensDigit():
    return (abs(n) // 10) % 10
More on functions

Built-in Functions

print(abs(-5))
print(max(2, 3))
print(min(2, 3))
print(pow(2, 3))
print(round(3.14))
print(round(3.14, 1))  # round with the given number of digits
print(type(5), end=" ")  <class 'int'>  <class 'str'>  <class 'bool'>
print(type(“hello”), end=" ")
print(type(True))

What other built-in functions are there?

See the official Python documentation.
More on functions

Variable scope

```python
def square(x):
    return x * x

def squareRoot(x):
    return x**0.5

def hypotenuse(a, b):
    return squareRoot(square(a) + square(b))
```

```
a = 3
b = 4
c = hypotenuse(a, b)
print("hypotenuse =", c)
```
More on functions

Variable scope

```python
def square(x):
    return x * x

def squareRoot(x):
    return x**0.5

def hypotenuse():
    return squareRoot(square(a) + square(b))

a = 3
b = 4
c = hypotenuse()
print("hypotenuse =", c)
```

Don’t do this!

In fact, never use globals!
def square(x):
    return x * x

def squareRoot(x):
    return x**0.5

def hypotenuse():
    a = 1
    return squareRoot(square(a) + square(b))

a = 3
b = 4
c = hypotenuse()
print("hypotenuse =", c)
def square(x):
    return x * x

def squareRoot(x):
    return x**0.5

def hypotenuse():
    global a
    a = 1
    return squareRoot(square(a) + square(b))

a = 3
b = 4
c = hypotenuse()
print("hypotenuse =", c)
Conditional Statements
Conditional Statements

3 Types:

if statement

if-else statement

if-elif-...-elif-else statement
If the expression evaluates to True:

- instruction1
- instruction2
- instruction3
- instruction4
- instruction5
If the expression evaluates to False:

- instruction1
- instruction2
- instruction5

If the expression evaluates to True or False:

- instruction3
- instruction4

Ideally, should evaluate to True or False.
1. **def** abs(n):
2. 	if(n < 0):
3. 		return -n
4. 	return n
If the first expression is true, we don’t skip checking the second one.

If both expressions evaluate to true:

```python
if(expression1):
    instruction3
    instruction4

if(expression2):
    instruction5
    instruction6
```

```python
instruction1
instruction2
instruction3
instruction4
instruction5
instruction6
instruction7
```
def message(age)
    if (age < 16):
        print("You can’t drive.")
    if (age < 18):
        print("You can’t vote.")
    if (age < 21):
        print("You can’t drink alcohol.")
    if (age >= 21):
        print("You can do anything that’s legal.")
print("Bye!")
if - else

instruction1
instruction2

if(expression):
  instruction3
  instruction4
else:
  instruction5
  instruction6

instruction7

If the expression evaluates to True.

instruction1
instruction2
instruction3
instruction4
instruction5
instruction6

Exactly one of the two blocks will get executed!
if - else

if (expression):
    instruction3
    instruction4
else:
    instruction5
    instruction6

instruction7

Exactly one of the two blocks will get executed!

If the expression evaluates to False.

instruction1
instruction2
instruction5
instruction6
instruction7
def f(x, y, z):
    if((x <= y and y <= z) or (x >= y and y >= z)):
        return True
    else:
        return False
def inOrder(x, y, z):
    if((x <= y and y <= z) or (x >= y and y >= z)):
        return True
    else:
        return False
def inOrder(x, y, z):
    if((x <= y and y <= z) or (x >= y and y >= z)):
        return True
    return False
What if you want to check 2 or more conditions?

```python
if(expression1):
    instruction1
else:
    if(expression2):
        instruction2
    else:
        instruction3
```

Only one of `instruction1`, `instruction2`, `instruction3` will be executed.
if (expression1):
    instruction1
else:
    if (expression2):
        instruction2
    else:
        instruction3
def numberOfQuadraticRoots(a, b, c):
    # Returns number of roots (zeros) of y = a*x**2 + b*x + c
    d = b**2 - 4*a*c
    if (d > 0):
        return 2
    elif (d == 0):
        return 1
    else:
        return 0
```python
def getGrade(score):
    if (score >= 90):
        grade = "A"
    elif (score >= 80):
        grade = "B"
    elif (score >= 70):
        grade = "C"
    elif (score >= 60):
        grade = "D"
    else:
        grade = "R"
    return grade
```
Some guidelines on correct usage of conditional statements

see notes
Practice Problem
Exercise: round(n)

Write a function that takes a float (or int) as input and returns the integer nearest to it.
Exercise: round(n)

Steps to follow

- Find a mental picture of the solution
- Write an algorithm
- Write the code
- TEST!
- Fix the bugs (if any)
Exercise: round(n)

- Find a mental picture of the solution

25.45

if >= 0.5, round up
- Find a mental picture of the solution

25.45

if \( \geq 0.5 \), round up

if \( < 0.5 \), round down
Exercise: round(n)

- Find a mental picture of the solution

25.45

if \( \geq 0.5 \), round up

if \( < 0.5 \), round down
Exercise: round(n)

- Find a mental picture of the solution

25.45

if >= 0.5, round up

if < 0.5, round down
Exercise: round(n)

- Find a mental picture of the solution

25.45

if \( \geq 0.5 \), round up \rightarrow \text{return } 25+1

if \( < 0.5 \), round down \rightarrow \text{return } 25
Exercise: round(n)

Steps to follow

- Find a mental picture of the solution
- Write an algorithm
- Write the code
- TEST!
- Fix the bugs (if any)
Exercise: round(n)

- Write an algorithm

25.45

if >= 0.5, round up \rightarrow \text{return } 25+1

if < 0.5, round down \rightarrow \text{return } 25

- Let n be the input number.
- Let intPart be the integer part of n.
  Let decPart be the decimal part of n.
- if decPart \geq 0.5, return intPart + 1
- if decPart < 0.5, return intPart
Exercise: round(n)

Steps to follow

- Find a mental picture of the solution
- Write an algorithm
- Write the code
- TEST!
- Fix the bugs (if any)
Exercise: round(n)

- Write the code

**algorithm:**
- Let n be the input number.
- Let intPart be the integer part of n.
  Let decPart be the decimal part of n.
- if decPart >= 0.5, return intPart + 1
- if decPart < 0.5, return intPart

```python
def round(n):
    intPart = int(n)
    decPart = n % 1
    if (decPart >= 0.5):
        return intPart + 1
    else:
        return intPart
```
Exercise: round(n)

- Find a mental picture of the solution
- Write an algorithm
- Write the code
- TEST!
- Fix the bugs (if any)
def testRound():
    assert(round(0) == 0)
    assert(round(0.5) == 1)
    assert(round(0.49999) == 0)
    assert(round(1238123.00001) == 1238123)
    assert(round(-0.5) == 0)
    assert(round(-0.49999) == 0)
    assert(round(-0.51) == -1)
    assert(round(-1238123.00001) == -1238123)
    print(“Passed all tests!”)
Exercise: round(n)

Steps to follow

- Find a mental picture of the solution
- Write an algorithm
- Write the code
- TEST!
- Fix the bugs (if any)
Exercise: `round(n)`

- Fix the bugs (if any)

Exercise for you.