CSCI 141
Computer Programming I

Filip Jagodzinski
Preliminaries: Announcements

• Lab hours
  • Check the CS support wiki (https://support.cs.wwu.edu/index.php/Main_Page) for lab schedules, instructions on how to log on remotely to a computer, etc.
  • There are 7 computer labs that are administered by CS in the CF building
  • If a lab/class is not scheduled for a lab when you want to do work, you can use that lab

• After hours
  • Labs are open 24 hours a day, 7 days a week when school is in session.
  • Access is restricted only by building hours and the auto-locking doors which are supposed to lock around 11:00pm
  • As long as students are in the building before it locks it's permissible for you to be in the labs as long as you need to (the infamous “all nighters”
Preliminaries: Announcements

Homework #1

- Has been posted to the course website
- Canvas has an auto-notification feature, so you can configure it to send you an email whenever a message is posted to the CSCI141 canvas page

TAs and Filip’s office hours, Tutoring

<table>
<thead>
<tr>
<th></th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>11am-noon</td>
<td>Filip, CF461</td>
<td>Filip, CF461</td>
<td>Filip, CF461</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2pm-4pm</td>
<td></td>
<td>Albert, CF405</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4pm-5pm</td>
<td>Robert, CF418</td>
<td></td>
<td>Robert, CF418</td>
<td>Robert, CF418</td>
<td>Robert, CF418</td>
</tr>
<tr>
<td>4pm-7pm</td>
<td>ACM Tutoring, CF162/164</td>
<td>ACM Tutoring, CF162/164</td>
<td>ACM Tutoring, CF162/164</td>
<td>ACM Tutoring, CF162/164</td>
<td>ACM Tutoring, CF162/164</td>
</tr>
</tbody>
</table>
From Last Time

Q: What is a function?

A. It is like a conjunction
B. It is the sequence of steps that are taken to solve a particular computation task
C. A procedure that is available for use in a programming language
D. A particular kind (or class) of a data item, such as String, int, or float
E. Often abstractly thought of as a “black box” with possible inputs and outputs
From Last Time

Q: What is a function?

A. It is like a conjunction
B. It is the sequence of steps that are taken to solve a particular computation task
C. A procedure that is available for use in a programming language
D. A particular kind (or class) of a data item, such as String, int, or float
E. Often abstractly thought of as a “black box” with possible inputs and outputs

syntax

type(16)  When executed  
<class ‘int’>
From Last Time

- `print(“Hello”, “Class”, 141)`
- `print(“Hello”, “Class”, 1, “4”, 1)`

Be sure you can determine the outputs of these functions “on paper” without relying on the Python interpreter.

- `int(float(“4.01”))` → “4.01” → int
- `float(“-44.55”)` → “-44.55” → float
- `float(78)` → 78 → float
- `str(1)` → 1 → str
- `str(78.34)` → 78.34 → str
From Last Time

```python
print("Hello", "Class", 141)  # Hello Class 141
print("Hello", "Class", 1, "4", 1)  # Hello Class 1 4 1

int(float("4.01"))  # 4
float("-44.55")  # -44.55
float(78)  # 78.0
str(1)  # '1'
str(78.34)  # '78.34'
```
Q: What is a variable?
Q: What are variables used for?
A variable is a name that refers to a value. Computers can store (remember) many values, hence a program can have many (millions, if the conditions are right), of variables.

```
myMomsHairColor = "blond"

myDogsName = "Emma"

myEmployer = "WWU"
```

My mom’s hair color ...  
My dog’s name ...  
I teach at ...
Today

Print
Variables
Operators
Operands
Strings
Memory
As we’ve already seen, `print` is a function that can accept one OR more inputs, separated by commas.

```
print("Hello World")
print("Hello Class", "141")
print("Hello", "Class", 1, "4", 1)
```

When `print` is invoked using multiple inputs, they are “printed” to the screen and concatenated. Q: **What is the default separation character between the multiple inputs for `print`?**
print

As we’ve already seen, print is a function that can accept one OR more inputs, separated by commas.

```
print("Hello World")
print("Hello Class", "141")
print("Hello", "Class", 1, "4", 1)
```

Hello World
Hello Class 141
Hello Class 1 4 1

When print is invoked using multiple inputs, they are “printed” to the screen and concatenated. Q: **What is the default separation character between the multiple inputs for print?**

Q: **Can we specify a different separation character (or characters)?**
As we’ve already seen, `print` is a function that can accept one OR more inputs, separated by commas.

```
print("Hello", "World", sep=",")
```

Q: Any intuitive guesses what `sep` accomplished?
As we’ve already seen, `print` is a function that can accept one OR more inputs, separated by commas.

```python
print("Hello", "World", sep=",")
```

Inputs 1 and 2
As we’ve already seen, `print` is a function that can accept one OR more inputs, separated by commas.

```python
print("Hello", "World", sep=" ")
```

This is an optional parameter that specifies the separation character if multiple inputs are supplied.

Q: What is the output of this statement?
As we’ve already seen, `print` is a function that can accept one OR more inputs, separated by commas.

```
print(“Hello”, “World”, sep=”,")  Hello,World
```
As we’ve already seen, `print` is a function that can accept one OR more inputs, separated by commas.

```python
print("Hello", "World", sep=',')
```

Hello,World

Q: Are there other optional parameters that we can use with the `print` function?
As we’ve already seen, the `print` function can accept one OR more inputs, separated by commas.

```
print("Hello", "World", sep=";")
```

```
Hello,World
```

The `end` parameter specifies what character(s) should be printed after all inputs have been printed. The default character is the newline character (which causes a “carriage return”).

```
print("Hello", end="a")
print("World")
```

Q: What is the output of the two lines of code on the left?
As we’ve already seen, `print` is a function that can accept one OR more inputs, separated by commas.

```
print(“Hello”, “World”, sep=“,”)
```

Hello,World

The `end` parameter specifies what character(s) should be printed after all inputs have been printed. The default character is the newline character (which causes a “carriage return”).

```
print(“Hello”, end=“a”)  
print(“World”)  
```

HelloaWorld
As we’ve already seen, `print` is a function that can accept one OR more inputs, separated by commas.

```python
def print("Hello", "World", sep=";")
```

The `end` parameter specifies what character(s) should be printed after all inputs have been printed. The default character is the newline character (which causes a “carriage return”).

```python
def print("Hello", end="a")
print("World")
```
Variables

As mentioned, “using” variables is a three step process ...

**Step 1**: Come up with a name for the variable
Variables

As mentioned, “using” variables is a three step process ...

**Step 1**: Come up with a name for the variable

- `numCows`
- `aMessage`
- `iAmAPalindrome`

These are three variable names ... they follow good convention because:

- Names are descriptive (not $x$, or $y$)
- Use camel capitalization

**Q**: When you declare a variable, what does that mean?
Variables

As mentioned, “using” variables is a three step process ...

**Step 1**: Come up with a name for the variable

```plaintext
numCows
aMessage
iAmAPalindrome
```

```plaintext
iAmAPalindrome
numCows
aMessage
```
Variables

As mentioned, “using” variables is a three step process ...

**Step 1**: Come up with a name for the variable

**Step 2**: Use the assignment operator

Think of an operator as a shorthand representation of a series of steps

```
numCows =
aMessage =
iAmAPalindrome =
```

```
iAmAPalindrome
numCows
```

```
aMessage
```
As mentioned, “using” variables is a three step process ...

**Step 1**: Come up with a name for the variable
**Step 2**: Use the assignment operator
**Step 3**: Initialize (or set the value of) the variable

```
numCows = 32
aMessage = “Hey Fred let us have coffee”
iAmAPalindrome = 3.232323
```

```
numCows

iAmAPalindrome

aMessage
```
As mentioned, “using” variables is a three step process ...

**Step 1**: Come up with a name for the variable  
**Step 2**: Use the assignment operator  
**Step 3**: Initialize (or set the value of) the variable

```plaintext
numCows = 32  
aMessage = “Hey Fred let us have coffee”  
iAmAPalindrome = 3.232323
```

When you initialize variables, you have a choice ... for example:

```plaintext
numCows = 32  
numCows = “32”
```

Q: Why choose one over the other?
As mentioned, “using” variables is a three step process ...

**Step 1**: Come up with a name for the variable

**Step 2**: Use the assignment operator

**Step 3**: Initialize (or set the value of) the variable

```plaintext
numCows = 32
aMessage = “Hey Fred let us have coffee”
iAmAPalindrome = 3.232323
```

Q: When you assign a value for a variable, what does that mean?
As mentioned, “using” variables is a three step process ...

**Step 1**: Come up with a name for the variable
**Step 2**: Use the assignment operator
**Step 3**: Initialize (or set the value of) the variable

```
numCows = 32
aMessage = "Hey Fred let us have coffee"
iAmAPalindrome = 3.232323
```

Each variable refers to an object
As mentioned, “using” variables is a three step process ...

**Step 1**: Come up with a name for the variable  
**Step 2**: Use the assignment operator  
**Step 3**: Initialize (or set the value of) the variable

```plaintext
numCows = 32
aMessage = “Hey Fred let us have coffee”
iAmAPalindrome = 3.232323
```

**Operators** work with **operands**
As mentioned, “using” variables is a three step process ...

**Step 1**: Come up with a name for the variable
**Step 2**: Use the assignment operator
**Step 3**: Initialize (or set the value of) the variable

```
numCows = 32
aMessage = "Hey Fred let us have coffee"
iAmAPalindrome = 3.232323
```

Operators work with operands

The assignment operator
Variables

As mentioned, “using” variables is a three step process ...

**Step 1**: Come up with a name for the variable
**Step 2**: Use the assignment operator
**Step 3**: Initialize (or set the value of) the variable

```
numCows = 32
aMessage = “Hey Fred let us have coffee”
iAmAPalindrome = 3.232323
```

**Operators work with operands**

The assignment operator requires two operands
Variables

As mentioned, “using” variables is a three step process ...

**Step 1**: Come up with a name for the variable
**Step 2**: Use the assignment operator
**Step 3**: Initialize (or set the value of) the variable

\[
\begin{align*}
\text{numCows} &= 32 \\
aMessage &= \text{"Hey Fred let us have coffee"} \\
iAmAPalindrome &= 3.232323
\end{align*}
\]

Operators work with operands

The assignment operator requires two operands

We say that the assignment operator is a binary operator
Statements and Expressions

```
myName = "Filip"  print(1+43)
```
Q: Is + a unary, binary, or ternary operator?
Statements and Expressions

**Statement**: an instruction that Python can “execute”

```
myName = "Filip"
```

```
print(1+43)
```
Statements and Expressions

**Statement**: an instruction that Python can “execute”

```python
myName = "Filip"
```

**Function**: a procedure that performs a certain task

```python
print(1+43)
```

**Q**: How do we know that `print` is a function?

**Inputs**: what is provided as input to the “black box” that is the `print` function

```python
print(44)
```

```text
44
```
**Statements and Expressions**

**Statement**: an instruction that Python can “execute”

```python
myName = "Filip"
```

**Expression**: A combination of values, variables, operators, calls to functions … etc.

```python
print(1+43)
```
Operators

There are other operators in addition to + and =

Q: What are the “basic” mathematical operators in python?
Operators

Some of these you are familiar with, but others (unless you’ve coded before) you are not

Q: What are these operators?

I don’t want you to “memorize” these. You’ll learn them by virtue of practice and using them for homework assignments.
Operators

Assignment, addition, subtraction, and multiplication ...

These “behave” just like they should ...
how you “know” of them from elementary mathematics

These four are all binary operators
Operators

The **division** operator is also “just like” the elementary math operator you are familiar with ... but unlike Java or several other programming languages, the division operator **ALWAYS** returns a result that is of type **float**.

```
7/3 = 2.3333333333
```
Operators

The **division** operator is also “just like” the elementary math operator you are familiar with ... but unlike Java or several other programming languages, the division operator **ALWAYS** returns a result that is of type **float**.

\[ \frac{7}{3} = 2.3333333333 \]

\[ \frac{6}{2} = \]

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>assignment</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>**</td>
<td>exponentiation</td>
</tr>
<tr>
<td>//</td>
<td>integer division</td>
</tr>
<tr>
<td>%</td>
<td>modulus</td>
</tr>
</tbody>
</table>
Operators

The **division** operator is also “just like” the elementary math operator you are familiar with … but unlike Java or several other programming languages, the division operator **ALWAYS** returns a result that is of type **float**.

\[
7/3 = 2.3333333333 \\
6/2 = 3.0
\]
Operators

= assignment
+ addition
- subtraction
* multiplication
/ division
** exponentiation
// integer division
%

The **exponentiation** operator evaluates an expression in which a **base** is raised to an **exponent**

Q: What is a **base** and what is an **exponent**?
Operators

= assignment
+ addition
- subtraction
* multiplication
/ division
** exponentiation
// integer division
%

The **exponentiation** operator evaluates an expression in which a **base** is raised to an **exponent**

\[ 2^4 = 2 \times 2 \times 2 \times 2 = 16 \]

\[ 2 \, ** \, 4 = 16 \]

base

exponent
Operators

= assignment
+
+ addition
-
- subtraction
*
* multiplication
/
/ division

** exponentiation
//
// integer division
%
%
modulus

The exponentiation operator evaluates an expression in which a base is raised to an exponent

\[ 2^4 = 2 \times 2 \times 2 \times 2 = 16 \]

\[ 2 \quad ** \quad 4 = 16 \]

\[ 2.0 \quad ** \quad 4 = 16 \]

Q: What format (integer, floating point, etc.) is the output when you use **?
Operators

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>assignment</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>**</td>
<td>exponentiation</td>
</tr>
<tr>
<td>//</td>
<td>integer division</td>
</tr>
<tr>
<td>%</td>
<td>modulus</td>
</tr>
</tbody>
</table>

The exponentiation operator evaluates an expression in which a base is raised to an exponent.

\[ 2^4 = 2 \times 2 \times 2 \times 2 = 16 \]

\[ 2.0 \times 2 \times 2 \times 2 = 16.0 \]

This is important because when you assign the value of a variable of the output of **, the “type” of that variable will be determined by python.
Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>assignment</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>**</td>
<td>exponentiation</td>
</tr>
<tr>
<td>//</td>
<td>integer division</td>
</tr>
<tr>
<td>%</td>
<td>modulus</td>
</tr>
</tbody>
</table>

The **exponentiation** operator evaluates an expression in which a **base** is raised to an **exponent**

```plaintext
aVar = 2**4
anotherVar = 2.0**4
```

**Q:** What are the data types for `aVar` and `anotherVar`?

**If you are not sure, what function should you use?**
Operators

= assignment
+ addition
- subtraction
* multiplication
/ division
** exponentiation
// integer division
%

The **exponentiation** operator evaluates an expression in which a **base** is raised to an **exponent**

\[
aVar = 2^{**4} \\
anotherVar = 2.0^{**4}
\]

```
type(aVar)  →  <class 'int'>
type(anotherVar)  →  <class 'float'>
```
The **integer division** operator performs division, and truncates ANY decimals. The type of the return is determined by the type(s) of the operands.

\[
\begin{align*}
7 \div 4 & \rightarrow 1 \\
5 \div 1 & \rightarrow 5 \\
5 \div 9 & \rightarrow ?
\end{align*}
\]

(on the board explanation)
Operators

The **integer division** operator performs division, and truncates ANY decimals. The type of the return is determined by the type(s) of the operands.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>assignment</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>**</td>
<td>exponentiation</td>
</tr>
<tr>
<td>//</td>
<td>integer division</td>
</tr>
<tr>
<td>%</td>
<td>modulus</td>
</tr>
</tbody>
</table>

Examples:

```
7 // 4 → 1
5 // 1 → 5
5 // 9 → 0
7.0 // 4 → ?
5 // 1.0 → ?
5.0 // 9.0 → ?
```
Operators

| =         | assignment |
| +         | addition   |
| -         | subtraction|
| *         | multiplication |
| /         | division |
| **        | exponentiation |
| //        | integer division |
| %         | modulus |

The **integer division** operator performs division, and truncates ANY decimals. The type of the return is determined by the type(s) of the operands.

\[
\begin{align*}
7.0 \div 4 & \rightarrow 1.0 \\
5.0 \div 1 & \rightarrow 5.0 \\
5.0 \div 9 & \rightarrow 0.0
\end{align*}
\]
Operators

|= assignment
|+ addition
|− subtraction
|* multiplication
|/ division
|** exponentiation
// integer division
|% modulus

The **modulus** operator performs long division and returns the “remainder”.

As you go on in computer science, the modulus operator is used very often due to data being “stored” as 1s and 0s.

\[
\begin{align*}
5 & \% 2 & \rightarrow & 1 \\
7 & \% 1 & \rightarrow & 0 \\
67 & \% 14 & \rightarrow & 7
\end{align*}
\]
The **modulus** operator performs long division and returns the “remainder”.

As you go on in computer science, the modulus operator is used very often due to data being “stored” as 1s and 0s.

\[
\begin{align*}
5 \mod 2 & \rightarrow 1 \\
7 \mod 1 & \rightarrow ? \\
67 \mod 14 & \rightarrow 
\end{align*}
\]
Operators

The **modulus** operator performs long division and returns the “remainder”.

As you go on in computer science, the modulus operator is used very often due to data being “stored” as 1s and 0s.

\[
\begin{align*}
5 \mod 2 &= 1 \\
7 \mod 1 &= 0 \\
67 \mod 14 &= ?
\end{align*}
\]

(on the board solution)
On the fly question

Q: What is the output of the following piece of code?

```
numCows = (9 % ( 5 // 1))
print(numCows)
```

A. -1  
B. 0   
C. 1   
D. 2   
E. 3   
F. 4   
G. 5   
H. None of the above
I. All of the above
On the fly question

Q: What is the output of the following piece of code?

```python
numCows = (9 % (5 // 1))
print(numCows)
```

A. -1
B. 0
C. 1
D. 2
E. 3
F. 4
G. 5
H. None of the above
I. All of the above

Inner most parentheses are executed first

5 divided by 1 using integer division = 5

Q: Now what is 9 modulus 5?
On the fly question

Q: What is the output of the following piece of code?

```python
numCows = (9 % ( 5 // 1))
print(numCows)
```

A. -1  
B. 0  
C. 1  
D. 2  
E. 3  
F. 4  
G. 5  
H. None of the above  
I. All of the above

Inner most parentheses are executed first

5 divided by 1 using integer division = 5

numCows = 4

Q: Now what is 9 modulus 5?

9 divided by 5 = 1 remainder 4
Q: What is the output of the following piece of code?

```
numCows = (9 % ( 5 // 1))
print(numCows)
```

The “input” to the `print` function is `numCows`, so Python will use the variable’s value as the input.

A. -1
B. 0
C. 1
D. 2
E. 3
F. 4
G. 5
H. None of the above
I. All of the above

numCows = 4
Variables

Q: Why are variables so important? Why/how are they used?
Variables

Q: Why are variables so important? Why/how are they used?

Humans are lousy at remembering things

Especially that pesky short-term memory

Q: What are the numbers we discussed last week?
Variables

Just like humans (assuming once you “remember” something), the value of a variable can be updated.

What I ask you the following question:

Q: What is your favorite color?

What “happens” in your brain?
Variables

Just like humans (assuming once you “remember” something), the value of a variable can be updated.

What I ask you the following question:

Q: What is your favorite color?

What “happens” in your brain?

You “go” to your place in your brain where you’ve chemically stored that value...

Q: Have you always had the same favorite color?
Variables

Just like humans (assuming once you “remember” something), the value of a variable can be updated

What I ask you the following question:

Q: What is your favorite color?

What “happens” in your brain?

You “go” to your place in your brain where you’ve chemically stored that value...

Q: Have you always had the same favorite color?

purple  yellow  green
Variables

Just like humans (assuming once you “remember” something), the value of a variable can be updated

The same functionality is provided by variables ... you can think of them as place holders of a value that you can update

```python
numCows = 32
print(numCows)
numCows = 44
print(numCows)
```
Variables

Just like humans (assuming once you “remember” something), the value of a variable can be updated

The same functionality is provided by variables ... you can think of them as place holders of a value that you can update

```python
numCows = 32
print(numCows)
numCows = 44
print(numCows)
```
Variables

Just like humans (assuming once you “remember” something), the value of a variable can be updated.

The same functionality is provided by variables ... you can think of them as place holders of a value that you can update.

The print method will “go” to the variable `numCows` and “get” the value from memory and print it.
Variables

Just like humans (assuming once you “remember” something), the value of a variable can be updated.

The same functionality is provided by variables ... you can think of them as place holders of a value that you can update.

```python
numCows = 32
print(numCows)
numCows = 44
print(numCows)
```

The value of `numCows` will be updated to now be 44. Technically (we’ll discuss this in much more detail later), a NEW object of type integer has been created, and the variable `numCows` will be reassigned to refer to the new object.
Variables

Just like humans (assuming once you “remember” something), the value of a variable can be updated.

The same functionality is provided by variables … you can think of them as place holders of a value that you can update.

```
numCows = 32
print(numCows)
numCows = 44
print(numCows)
```

So on the subsequent print, the value printed will be 44.
More on variables
Base 10 / binary conversion