Hello, World!

Camp Inc. Coding Track

Summer 2016

Don't just buy a new video game, make one.
Don't just download the latest app, help design it.
Don't just play on your phone, program it.
No one is born a computer scientist, but with a little hard work and some math and science, just about anyone can become one.

— Barack Obama

Reading a Book

Programs are read from top to bottom, following the steps in order. Consider the following procedure to read a book:

- Open book
- While not at end of book:
  - Read visible pages
  - Flip to next page
  - If tired:
    - Put bookmark in current position
    - Close book
    - Sleep
    - Open book to current position and remove bookmark
- Close book

Reading a Book

How do we know we are tired? Lets add a counter, and rest every 10 times we complete the loop

- Open book
- While not at end of book:
  - Read visible pages
  - Flip to next page
  - If tired:
    - Put bookmark in current position
    - Close book
    - Sleep
    - Open book to current position and remove bookmark
- Close book

Reading a Book

How do we know we are tired? Lets add a counter, and rest every 10 times we complete the loop

- Open book
- Let x be 0
- While not at end of book:
  - Let x be $x + 1$
  - Read visible pages
  - Flip to next page
  - If x is 10:
    - Put bookmark in current position
    - Close book
    - Sleep
    - Open book to current position and remove bookmark
    - Let x be 0
- Close book

Your First Python Program

Open the file `hello.py` and type the following contents:

```
print("Hello, World!")
```

Your First Python Program

Open the file `hello.py` and type the following contents:

```
print("Hello, World!")
```

- `print` is a **function**, it takes arguments, does things (in this case, print to our console), and returns things

Your First Python Program

Open the file `hello.py` and type the following contents:

```
print("Hello, World!")
```

- `print` is a **function**, it takes arguments, does things (in this case, print to our console), and returns things
- The **arguments** to our `print` call is just the **string** `"Hello, World"`.

Your First Python Program

Open the file `hello.py` and type the following contents:

```
print("Hello, World!")
```

- `print` is a **function**, it takes arguments, does things (in this case, print to our console), and returns things
- The **arguments** to our `print` call is just the **string** `"Hello, World"`.
- A string can be written with either single or double quotes, so this program does the same:

```
print('Hello, World!')
```

Variables

In math, when we write $=$, we mean **relation**:

$$x^3 - x + 1 = 0$$

Variables

In math, when we write $=$, we mean **relation**:

$$x^3 - x + 1 = 0$$

This is **not** the same in programming! In most programming languages, including Python, = means **assignment**. We let the *variable on the left* take the *value on the right*.

```
x = 10
y = x + 2
x = 11
print(x, y)
```

Variables

In math, when we write $=$, we mean **relation**:

$$x^3 - x + 1 = 0$$

This is **not** the same in programming! In most programming languages, including Python, = means **assignment**. We let the *variable on the left* take the *value on the right*.

```
x = 10
y = x + 2
x = 11
print(x, y)
```

```
11 12
```

Accepting User Input

The `input` function takes a prompt string, and returns the string the user types.

```
name = input('What is your name? ')
print('Nice to meet you', name)
```

Simple Operators

Python provides a simple notation to write mathematical statements.

```
print(5+11, 18-2, 2*8, 4**2, 32/2, 33//2, 56%20)
```

+	Addition	/	Division
-	Subtraction	//	Integer Division
*	Multiplicaton	%	Modulus (division remainder)
**	Exponentation		

The Interactive Interpreter

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
```

The Interactive Interpreter

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0

>>> 17 / 3    # / division returns a float
5.666666666667
>>> 17 // 3   # // division returns an int
5
>>> 17 % 3    # % returns the remainder of the division
2
>>> 5 * 3 + 2 # quotient * divisor + remainder = dividend
17
```

Variables

```
>>> 17 / 3    # / division returns a float
5.666666666667
>>> 17 // 3   # // division returns an int
5
>>> 17 % 3    # % returns the remainder of the division
2
>>> 5 * 3 + 2 # quotient * divisor + remainder = dividend
17

>>> dividend = 17
>>> divisor = 3
>>> quotient = dividend // divisor
>>> remainder = dividend % divisor
>>> quotient * divisor + remainder == dividend
True
```

Another Kind of Equals

`==` is for **equality testing**. An expression `a == b` will be `True` iff `a` has the same value as `b`, `False` otherwise.

Another Kind of Equals

`==` is for **equality testing**. An expression `a == b` will be `True` iff `a` has the same value as `b`, `False` otherwise.

```
name = input('What is your name? ')
print('It is', name == 'Jack',
      'that your name is the best.')
```

Branching

```
if condition:
    # do something
elif other_condition:
    # do something else
else:
    # do something else
```

Branching

```
if condition:
    # do something
elif other_condition:
    # do something else
else:
    # do something else
```

```
name = input('What is your name? ')
if name == 'Jack':
    print('Your name is the best!')
elif len(name) == 4:
    print('Your name is 4 letters!')
else:
    print('Pleased to meet you', name)
```

More Comparison Operators

`<`	Less than	`<=`	Less or equal	`!=`	Not equal
`>`	Greater than	`>=`	Greater or equal	`is`	Same instance

More Comparison Operators

`<`	Less than	`<=`	Less or equal	`!=`	Not equal
`>`	Greater than	`>=`	Greater or equal	`is`	Same instance

```
age = int(input('What is your age? '))
if age < 0:
    print('I don\'t think so')
elif age <= 10:
    print('Wow! You\'re young!')
elif age != 16:
    print('Cool cool.')
else:
    print('Sweet sixteen.')
```

Indentation Denotes Scope

In Python, indentation not only provides style to help yourself and others read your code, but also provides functionality by **denoting the scope of the operation**. Consider the following example:

```
# i was defined previously in this program
if i > 0:
    print("i is positive")
    if i % 2 == 0:
        print("i is even")
    print("hello")
print("goodbye")
```

Indentation Denotes Scope

In Python, indentation not only provides style to help yourself and others read your code, but also provides functionality by **denoting the scope of the operation**. Consider the following example:

```
# i was defined previously in this program
if i > 0:
    print("i is positive")
    if i % 2 == 0:
        print("i is even")
    print("hello")
print("goodbye")
```

1. What will be printed if `i` is `3`
2. What will be printed if `i` is `-2`
3. What will be printed if `i` is `4`