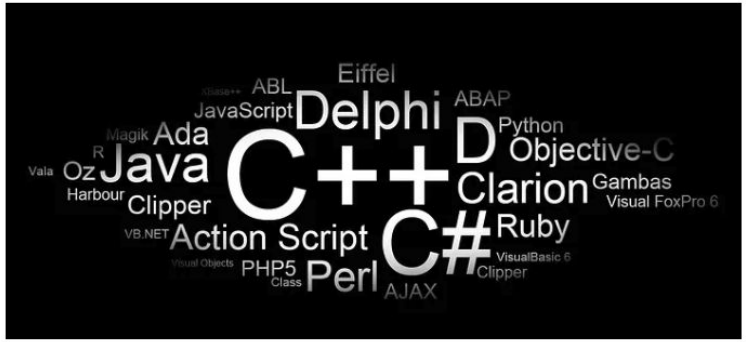
Python Basics

Programming Languages for Game Development

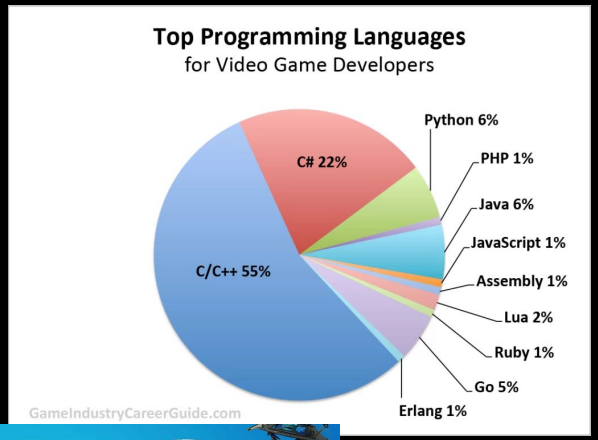
The language of video games

Jon Mohon · Follow · 3 min read · Mar 25, 2019

"Why are games written in C++?" A friend of mine asked me this recently and I did a bit of research which I thought I would share.



Unity



So… why Python?

Python is widely considered as the most beginner-friendly programming language.

- Excellent for learning basic programming concepts
- You can use it to quickly prototype a computer game (lightweight, 2D)

And, we (sadly) only have less than three weeks for this course…

Variables…

Variables

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

```
x = 2026
y = 'UChicago'
```

Variables

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

```
x = 2026
y = 'UChicago'
x = 12.5
```

x

What value do you think x would return now?

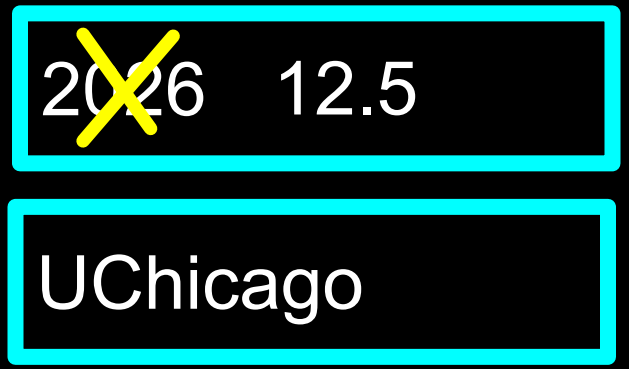
y

UChicago

Variables

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

```
x = 2026
y = 'UChicago'
x = 12.5
```



Constants

- Fixed values such as numbers, letters, and strings, are called “constants” because their value does not change
- Numeric constants are as you expect
- String constants use single quotes (') or double quotes (")

```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hello world')
Hello world
```

Reserved Words

You cannot use reserved words as variable names / identifiers

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

Python Variable Name Rules

- Must start with a letter or underscore _
- Must consist of letters, numbers, and underscores
- Case Sensitive

```
Good:      spam     eggs    spam23      _speed
Bad:       23spam       #sign   var.12
Different:      spam    Spam    SPAM
```

Naming Variables

Since we programmers are given a choice in how we choose our variable names, there is a bit of “best practice”

```
x1q3z9ocd = 40.0
x1q3z9afd = 32.0
x1q3p9afd = (x1q3z9ocd - x1q3z9afd) * 5 / 9
print(x1q3p9afd)
```

What is this bit of code doing?

Naming Variables

Since we programmers are given a choice in how we choose our variable names, there is a bit of “best practice”

```
x1q3z9ocd = 40.0
x1q3z9afd = 32.0
x1q3p9afd = (x1q3z9ocd - x1q3z9afd) * 5 / 9
print(x1q3p9afd)
```

```
a = 40.0
b = 32.0
c = (a - b) * 5 / 9
print(c)
```

What is this bit of code doing?

Naming Variables

Since we programmers are given a choice in how we choose our variable names, there is a bit of “best practice”

```
x1q3z9ocd = 40.0
x1q3z9afd = 32.0
x1q3p9afd = (x1q3z9ocd - x1q3z9afd) * 5 / 9
print(x1q3p9afd)
```

```
a = 40.0
b = 32.0
c = (a - b) * 5 / 9
print(c)
```

What is this bit of code doing?

```
fahrenheit = 40.0
offset = 32.0
celsius = (fahrenheit - offset) * 5 / 9
print(celsius)
```

Assignment Statements

- We assign a value to a variable using the assignment statement (=)
- An assignment statement consists of an expression on the right-hand side and a variable to store the result

```
x = 3.9 * x * ( 1 - x )
```

Expressions…

Numeric Expressions

- Because of the lack of mathematical symbols on computer keyboards - we use “computer-speak” to express the classic math operations
- Asterisk is multiplication
- Exponentiation (raise to a power) looks different than in math

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Numeric Expressions

```
x = 1 + 2 * 3 - 4 / 5 ** 6
```

$$x = 1 + 2 \times 3 - \frac{4}{5^6}$$

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Order of Operations

Highest precedence rule to lowest precedence rule:

- Parentheses are always respected
- Exponentiation (raise to a power)
- Multiplication, Division, and Remainder
- Addition and Subtraction
- Left to right

Parenthesis
Power
Multiplication
Addition
Left to Right

** similar to standard mathematical order of operations*

“Best practice”

When writing code,

- keep math expressions simple enough that they are easy to understand
- break long series of math operations up to make them more clear

Parenthesis
Power
Multiplication
Addition
Left to Right

Printing output…

Data Types…

What Does “Type” Mean?

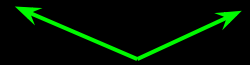
- In Python, the values that are assigned to variables are always of a certain “type”.

- We can distinguish three general types:
  - Strings
  - Numbers
  - Boolean values (`True` or `False`)

What Does “Type” Mean?

- In Python, the values that are assigned to variables are always of a certain “type”.

- We can distinguish three general types:
  - Strings
  - Numbers
  - Boolean values (`True` or `False`)



CASE SENSITIVE!!!

`TRUE`

`false`

Type Matters

- Python knows what “type” everything is
- We can ask Python what type something is by using the type() function

```
>>> eee = 'hello ' + 'there'
>>> type(eee)
<class'str'>
>>> type('hello')
<class'str'>
>>> type(1)
<class'int'>
>>> eee = eee + 1
```

Type Matters

- Python knows what “type” everything is
- We can ask Python what type something is by using the type() function
- Some operations are prohibited
- You cannot “add 1” to a string

```
>>> eee = 'hello ' + 'there'
>>> type(eee)
<class'str'>
>>> type('hello')
<class'str'>
>>> type(1)
<class'int'>
>>> eee = eee + 1
Traceback (most recent call
last):
File "<stdin>", line 1, in
<module>
TypeError: can only concatenate
str (not "int") to str
```

Several Types of Numbers

- Numbers have two main types
  - Integers are whole numbers: -14, -2, 0, 1, 100, 401233
  - Floating Point Numbers have decimal parts: -2.5 , 0.0, 98.6, 14.0
- There are other number types - they are variations on float and integer

```
>>> xx = 1
>>> type (xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class'float'>
>>>
```

Type Conversions

- When you put an integer and floating point in an expression, the integer is implicitly converted to a float
- You can control this with the built-in functions int() and float()

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class'float'>
>>>
```

Integer Division

Integer division produces a floating point result

```
>>> print(10 / 2)
5.0
>>> print(9 / 2)
4.5
>>> print(99 / 100)
0.99
>>> print(10.0 / 2.0)
5.0
>>> print(99.0 / 100.0)
0.99
```

This was different in Python 2.x

String Conversions

- You can also use int() and float() to convert between strings and integers
- You will get an error if the string does not contain numeric characters

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str
(not "int") to str
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
with base 10: 'x'
```

Comments in Python

- Anything after a # is ignored by Python
- Why comment?
  - Describe what is going to happen in a sequence of code
  - Document who wrote the code or other ancillary information
  - Turn off a line of code - perhaps temporarily

```
# Get the name of the file and open it
name = input('Enter file:')
handle = open(name, 'r')

# Count word frequency
counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) +
1

# Find the most common word
bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count >
bigcount:
        bigword = word
        bigcount = count

# All done
print(bigword, bigcount)
```

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

Adapted by Minh Tran, University of Chicago, Department of Computer Science for use in the pre-college course Creating Computer Games for Learning (Summer 2026).

...