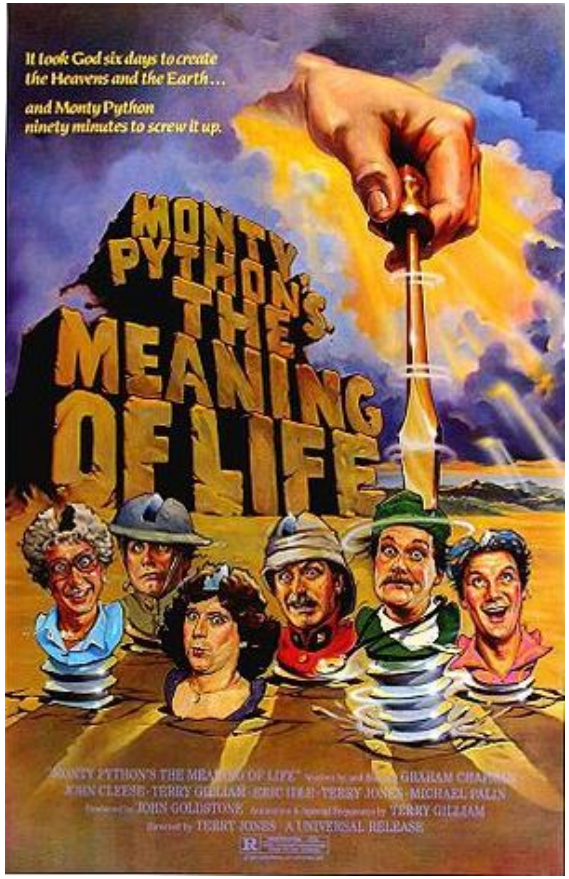
Python Basics

- Values, Types, Variables, Expressions
- Assignments
- I/O
- Control Structures



Values, Types and Variables

- 1, 2, 3.14 and 'Hello, World!' are values
- Values belong to types: int, float, str, bool …
- If you aren't sure what type something is you can ask

```
>>> type (3.2)  → <type 'float'>
>>> type (True) → <type 'bool'>
```

- An assignment statement creates new variables at run-time

```
>>> n = 17
>>> type (n)  → <type 'int'>
```

- A variable name starts with letter, followed by 0 or more letters, digits or _
  - Choose an incorrect variable name, get error

```
>>>  more@ = 1000 → Syntax error: invalid syntax
```

31 Reserved Words

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

Statements

- Unit of code the interpreter executes and if there is a result, shows it
- Seen the `print` and the `assignment`
- Several statements are a script

```
>>> print 1
>>> x =2
>>> print x
```

- Produces the output

```
1
2
```

Operators

- Operators are special symbols representing operations (such as, addition, multiplication, etc.)

  `+  -  *  /  **` (exponentiation)

- Values they operate on are called operands
- If both operands are integers, performs integer operation in Python 2.7

```
>>> minutes = 59
>>>print minutes / 60
0
```

- In Python 3.0 `//` is integer division, otherwise does floating point division
- Modulus operator `7 % 3` → `1`

Expressions

- An expression is a combination of values, operands and operators
- If you type an expression in interactive mode, it will show you the result

```
>>> 1 + 1
2
```

- Order of operations PEMDAS

String Operations

- `+` is concatenation

```
>>> print  '100' + '150'
100150
```

- Python has a "string format" operator `%`. This functions analogous to `printf` format strings in `C`

```
"foo=%s bar=%d" % ("blah", 2) → "foo=blah bar=2"
```

- In Python 3 and 2.6+, this was supplemented by the `format()` method of the `str` class

```
"foo={0} bar={1}".format("blah", 2).
```

Input

- `raw_input` gets input from keyboard

```
>>> name = raw_input ('What is your name?\n')
Chuck
>>> print name
Chuck
>>> speed = raw_input('How fast were you going?\n)
23
>>>  int (speed)
>>> print speed + 5
28
```

- If user types in a non-integer → ValueError: invalid literal for int( )
- In Python 3.0 this function is input

Comments

- `#compute the percentage of the hour that has elapsed`
- `percentage = int ( (float(minutes) / 60) * 100)`
- Or
- `percentage = int ((float (minutes)/60) * 100 )    #percentage of hour`

- Don't do this!
- `v = 5  #assign 5 to v`
- Better – use a good variable name
- `velocity = 5`

Boolean Expressions

- Expression that is either true or false
- Comparison operator `==` `!=` `<` `>` `<=` `>=` `is` and `is not`
- Three logical operators `and` `or` and `not`

```
>>>not (True)  → False
```

- `0` is interpreted as `false`
- Any nonzero number is interpreted as `True`

```
>>> 17 and True → True
```

- Comparisons may be chained, for example `a <= b <= c`

Conditional statements

```
 if x > 0:
    print ‘x is positive’j     #NOTE indentation!!!

if x % 2 = 0:
    print ‘x is even’
else:
    print ‘x is odd’

if x > 0:
    print ‘x is positive’
elif x < 0:                          #NOTE elif
    print ‘x is negative’
else
    print ‘x is zero’
```

Nested Conditionals

- Indentation is IMPORTANT

```
if x == y:
  print 'x is same as y'
else:
   if x < y:
      print 'x is less than y'
   else:
      print 'x is greater than y'
```

Nested Conditionals

- Indentation is IMPORTANT

```
if x == y:
  print 'x is same as y'
else:
   if x < y:
      print 'x is less than y'
else:
   print 'x is greater than y'
```

- Gives an error

```
IndentationError: unindent does not match any outer indentation level
```

Error Conditionals - try/catch

```
strtemp = input ('Enter Fahrenheit temperature: ')
try:
    fahr = float (strtemp)
    cel = (fahr - 32.0) * 5.0 /9.0
    print cel
except:
    print ('Please enter a number')
```

Short Circuiting

- Expressions are evaluated left to right
- If nothing to be gained from evaluating further (because value already known), it short circuits

```
if num != 0  and sum / num > 0 :
      print (sum / num)
else:
     print ('Cannot do division')
```

switch

- Python doesn’t have one
- Can fake with an array or dictionary or lambdas
- More on that later