

# Python Basics

Functions, control, logic

## CS101 Lecture #5

# Administrivia

# Administrivia

- Homework #2 was due Oct 3<sup>rd</sup>.
- Don't send emails for homework deadline extension, except for *Very* special conditions
- Labs this Wednesday

# Warmup (review)

# Function

```
def f(x):  
    y = x ** 2  
    area = 0.5 * math.pi * y  
    return area
```

# Functions cont.d

# Example

```
def greetings():  
    print('Bom dia!')  
    print('Bonjour!')  
    print('Hello')  
    print('Ni Hao!')  
    print('Shalom!')  
    print('Guten tag!')  
    print('Konichiwa!')  
    print('As-salamu alaykum!')
```

header

body

# Example

```
def pow(a, b):  
    y = a ** b  
    return y
```



# Defining a function

- We define a function with the following
  - Keyword `def`
  - The name of the function
  - A pair of parentheses
  - Arguments inside the parentheses (optional)
  - Return value (optional)
  - A **block** of code

```
def pow(a, b):  
    y = a ** b  
    return y
```

# Block

- A section of code grouped together
- Starts with a :
- Contents of the block are ***indented at the same level***

```
def pow(a, b) :  
    y = a ** b  
    return y
```

# *Example*

```
a = 5
def fun():
    a = 3
    print(a)

fun(a)
```

# *Example*

```
a = 5
def fun():
    a = 3
    print(a)

fun(a)
```

# Scope

- Variables defined inside of a block are *Independent* of variables outside of the block
- Variables inside a block do not exist outside of the block – scope
- The scope of a function is isolated from the rest of the code

```
def pow(a, b) :  
    y = a ** b  
    return y
```

# Example

```
def fun() :  
    a = 3  
    b = 4  
    a = a + b  
fun()  
print(a)
```

# Example

```
a = 5
def fun():

    b = 4
    a = a + b
fun()
print(a)
```

# Example

```
a = 5
def fun():
    a = 3
    b = 4
    a = a + b
fun()
print(a)
```



# *return*

```
a = 5
def fun():
    a = 3
    b = 4
    a = a + b
    return a
a = fun()
print(a)
```

# *return*

- Functions can return values to the *outer* scope with the keyword **return**
- The returned values can be assigned to a variable after the function call

# *return*

```
a = 5
def fun():
    a = 3
    b = 4
    a = a + b
    return a
fun()
print(a)
```

# Example

```
a = 5  
def fun():  
    a = 3  
    return a
```

```
b = fun()  
print(a)  
print(b)
```

# *return*

- Does the code below face an error?
- Does the print statement take effect if invoking the function?

```
def three():  
    return 3  
    print('3')
```

# Arguments

- Functions can accept values as argument (input, parameters)
- These variables are declared in the function header
- Multiple arguments are separated by commas

```
def print_message(msg) :  
    print(msg)
```

# Example

```
def fun(a):  
    return a+2
```

```
x = fun(2) * fun(3+1)
```

What is the value of x?

# Example

```
def fun(a):  
    return m.title().swapcase()
```

```
x = fun('abc') * fun('acab')
```

What is the value of x?

- A 'AbbAcab'
- B 'aBBaCAB'
- C 'abbacab'
- D 'ABBACAB'



# Example

```
def fun(a, b):  
    c = ((a+' ')*len(b)).title()
```

```
x = fun('ab', 'caa')
```

What is the value of x?

A 'ab ab ab'

B 'Ab Ab Ab'

C 'AB AB AB'

D None of the above

# *Return value as argument*

- The returned value of a function call can be passed to another function as argument

```
def three():  
    return 3
```

```
a = pow(three(), three()+2)
```

# Example

```
def fun(a):  
    return a+2
```

```
x = fun(2) * fun(fun(2))
```

What is the value of x?

# Conditional Execution

# Control flow

- A simple program execution flow

```
1. def pow(a, b):  
2.     y = a** b  
3.     return y  
4.  
5. a = 2  
6. b = -3  
7. print(pow(a, b))
```



# Control flow

- We want more flexible control of the program execution flow logic than this!

```
1. def pow(a, b):  
2.     y = a** b  
3.     return y  
4.  
5. a = 2  
6. b = -3  
7. print(pow(a, b))
```

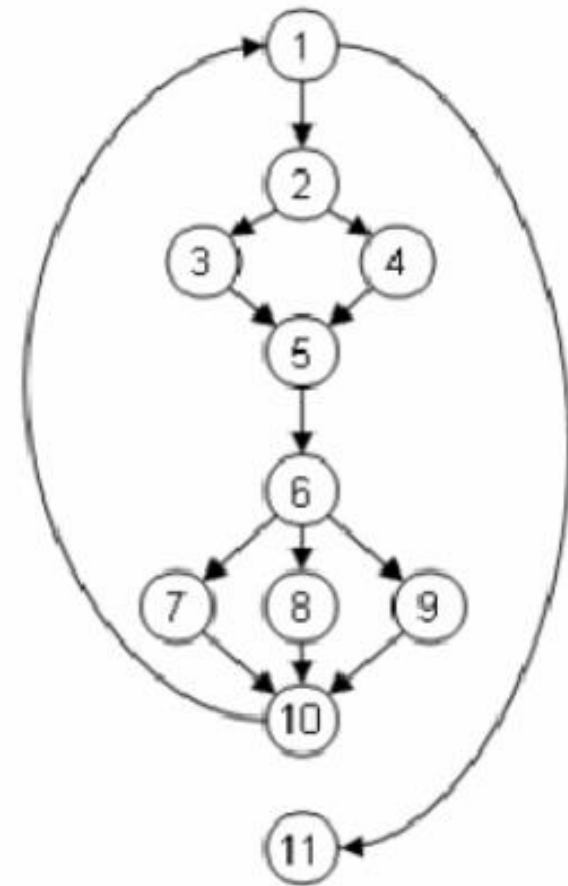


# Control flow

- We want more flexible control of the program execution flow logic than this!
- ***Control flow*** allows us to define it

# Control flow

Node	Statement
(1)	while(x<100){
(2)	if (a[x] % 2 == 0) {
(3)	parity = 0;
	}
(4)	else {
(5)	parity = 1;
(6)	}
(6)	switch(parity){
	case 0:
(7)	println( "a[" + i + "] is even");
	case 1:
(8)	println( "a[" + i + "] is odd");
	default:
(9)	println( "Unexpected error");
	}
(10)	x++;
(11)	}
	p = true;



A more expressive control flow graph (Code in C language; not required)



# *Conditional flow: if statement*

- Conditional flow allows you to execute (or not) a block of code based on logical comparison

# *Example: if statement*

```
ans = input('Enter a number: ')\nif float(ans) < 0:\n    print('the input number is negative.')
```

# *if statement*

- A `if` statement has the following:
  - The keyword `if`
  - A logical comparison (result in a **boolean** type)
  - A block of code (starts with a `:` )

# *if statement*

- Let us make decisions during the program execution
- Change program behavior based on different execution conditions

# *Example: if statement*

```
ans = input('Enter a number: ')\nif float(ans) < 0:\n    print('the input number is negative.')\nif float(ans) >= 0:\n    print('the input number is positive or zero.')
```

# *Example: if-else statement*

```
ans = input('Enter a number: ')\nif float(ans) < 0:\n    print('the input number is negative.')\nelse:\n    print('the input number is positive or zero.')
```

- For two conditional branches that are logically complementary

# *Example: if-else statement*

```
ans = input('Enter a number: ')\nif float(ans) < 0:\n    print('the input number is negative. ')\nif float(ans) > 0:\n    print('the input number is positive. ')\nif float(ans) == 0:\n    print('the input number is zero.')
```

# *Example: if-else statement*

```
ans = input('Enter a number: ')
if float(ans) < 0:
    print('the input number is negative.')
elif float(ans) > 0:
    print('the input number is positive.')
else:
    print('the input number is zero.')
```

- For multiple conditional branches that are logically complementary
- `elif` means “else if”



# Control flow

- More control flow syntaxes to come!

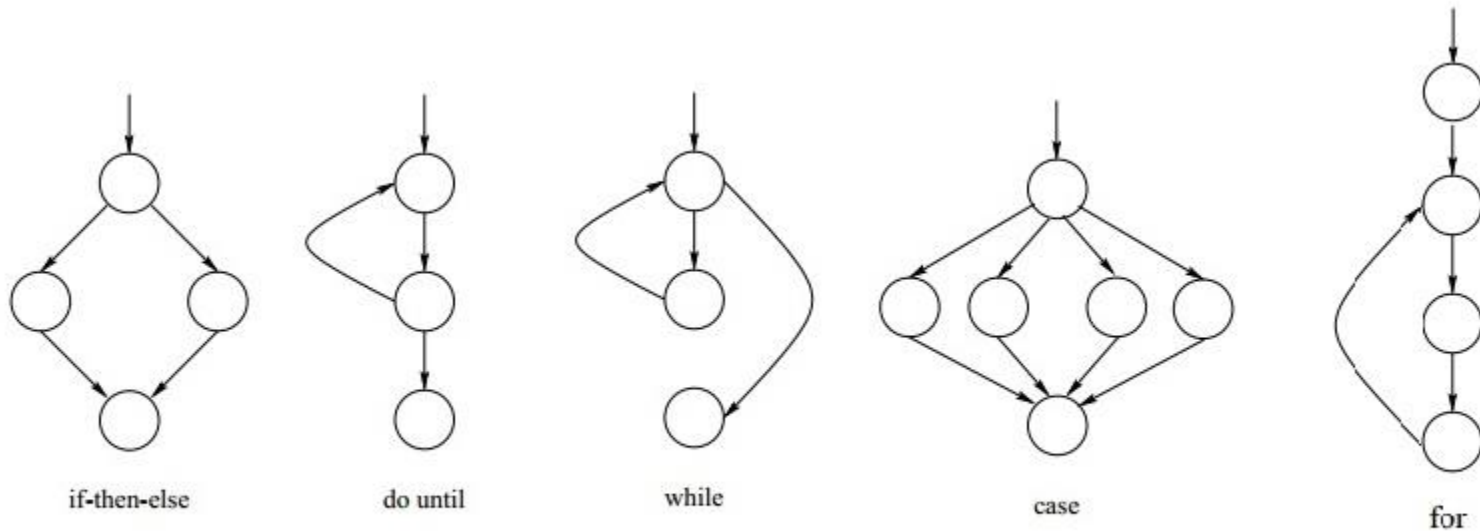


Figure 1: Flow graph representation.

# *Control flow*

If they have eggs, get six!

<http://www.dslreports.com/forum/r25743814-Wife-of-a-computer-programmer>

# Boolean Logic

# Boolean

- `Bool` is a data type with two possible values
  - `True`
  - `False`
- We use these to make decisions
- Their logic is based on Boolean algebra
- Operators
  - `and` (`&`)
  - `or` (`|`)
  - `not`

# *Example: Boolean logic*

$x > 0$

$(x > 0) \text{ or } (x < -10)$

$(x > 0) \text{ and } (x \leq 10)$

$0 < x \leq 10$

# Boolean

- Assign a boolean operation to a variable

```
x = 5
y = (x < 0) or (x > -2)
type (x)
type (y)
print(y)
```

# *Boolean operators (review)*

<b>and</b>	True	False
True	<b>True</b>	False
False	False	False

True when BOTH inputs are true

<b>or</b>	True	False
True	True	True
False	True	<b>False</b>

True when EITHER input is true

# *Boolean operators*

<b>not</b>	result
True	<b>False</b>
False	True

Inverts input value



# Example

```
def fun():  
    return True and False
```

```
x = fun() and not (True or False)
```

What is the value of x?

A True

B False

# Comparison operators

- Comparison operators also produces **bool** type
  - Less than, <
  - Greater than, >
  - Less than or equal to, <=
  - Greater than or equal to, >=
  - Equal to, ==
  - Not equal to, !=

# Example

a = 5

b = 3

x = (a < 5) or ((b <= 3) and (a != b))

What is the value of x?

A True

B False

# Example

```
a = 5
```

```
b = 'hello world!'
```

```
x = (a < 5) or (b[len(b)] == '!')
```

What is the value of x?

A True

B False

# Example

```
a = 5
```

```
b = 'hello world!'
```

```
x = (a < 5) and (b[len(b)] == '!')
```

What is the value of x?

A True

B False

# *Try out this..*

```
a = 5  
b = 'hello world!'  
  
x = (a < 5) & (b[len(b)] == '!')
```

# Reminder

# *Reminder*